Volume 20 Number 2

ACTA CYBERNETICA

Szeged, 2011

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. An electronic version of the puplished paper is provided for the authors in PDF format.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

Web access. The above informations along with the contents of past issues are available at the Acta Cybernetica homepage http://www.inf.u-szeged.hu/actacybernetica/ .

EDITORIAL BOARD

Ferenc Gécseg
Department of Computer Algorithms
and Artificial Intelligence
University of Szeged
Szeged, Hungary
gecseg@inf.u-szeged.hu

Jozef Gruska
Institute of Informatics/Mathematics
Slovak Academy of Science
Bratislava, Slovakia
gruska@savba.sk

Tibor Gyimóthy
Department of Software Engineering
University of Szeged
Szeged, Hungary
gyimothy@inf.u-szeged.hu

Helmut Jürgensen
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
helmut@csd.uwo.ca

Zoltan Kato
Department of Image Processing
and Computer Graphics
Szeged, Hungary
kato@inf.u-szeged.hu

Alice Kelemenová
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

László Lovász
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

Gheorghe Păun
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

András Prékopa
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

Arto Salomaa
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

László Varga
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

Heiko Vogler
Department of Computer Science
Dresden University of Technology
Dresden, Germany
Heiko.Vogler@tu-dresden.de

Gerhard J. Woeginger
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

Weighted Automata: Theory and Applications

Guest Editors:

Manfred Droste
Institut für Informatik
Universität Leipzig
Germany
droste@informatik.uni-leipzig.de

Heiko Vogler
Fakultät Informatik
Technische Universität Dresden
Germany
Heiko.Vogler@tu-dresden.de

Preface

This special issue contains papers on topics of the workshop

"Weighted Automata: Theory and Applications (WATA 2010)"

which took place at the Universität Leipzig, Germany, from May 3-7, 2010.

As for its predecessors WATA 2002, WATA 2004, WATA 2006, and WATA 2008, the goal of this workshop was to highlight the field of weighted automata, ranging from the theory of formal power series and probabilistic logics to applications for real-time systems and natural language processing.

The workshop was attended by 48 participants from 11 countries. Three tutorials were given by

Paul Gastin (Cachan, France),
Kim Larsen (Aalborg, Denmark),
Mark-Jan Nederhof (St. Andrews, UK).

In addition, six invited lectures were presented by

Christel Baier (Dresden, Germany),
Patricia Bouyer-Decitre (Cachan, France),
Miroslav Ćirić (Niš, Serbia),
Zoltán Ésik (Szeged, Hungary),
Dietrich Kuske (Bordeaux, France),
Andreas Maletti (Tarragona, Spain).

Furthermore, 21 talks were selected as contributed communications.

This workshop was financially supported by the ESF activity "Automata: from Mathematics to Applications (AutoMathA)" and by "Vereinigung von Förderern und Freunden der Universität Leipzig e.V.".

After the workshop, a call for papers for a special issue of "Acta Cybernetica" on "Weighted Automata: Theory and Applications" was issued. Following the standard refereeing procedure, we were pleased to accept the present two papers for this special issue.

Manfred Droste (Leipzig)
Heiko Vogler (Dresden)

October 2011

Acta Cybernetica **20** (2011) 211–221.

The Support of a Recognizable Series over a Zero-sum Free, Commutative Semiring is Recognizable[†]

Daniel Kirsten[‡]

Abstract

We show that the support of a recognizable series over a zero-sum free, commutative semiring is a recognizable language. We also give a sufficient and necessary condition for the existence of an effective transformation of a weighted automaton recognizing a series S over a zero-sum free, commutative semiring into an automaton recognizing the support of S.

Keywords: weighted automata, recognizable series, support

1 Introduction

One stream in the rich theory of formal power series deals with connections to formal languages. To each formal power series, one associates a certain language, called the support, which consists of all words which are not mapped to zero.

It is well-known that the support of a recognizable series is not necessarily a recognizable language. However, for large classes of semirings, it is known that the support of a recognizable series is always recognizable, see [3, 5, 9] for recent overviews. These classes include all positive semirings (semirings which are both zero-divisor free and zero-sum free), all finite, and more generally, all locally finite semirings.

WANG introduced the notion of a quasi-positive semiring (that is, for every $k \in \mathbb{K} \setminus \{0\}$, $\ell \in \mathbb{K}$, $n \in \mathbb{N}$, we have $k^n + \ell \neq 0$), and showed that the support of a recognizable series over a commutative, quasi-positive semiring is always a recognizable language [11]. Every quasi-positive semiring is zero-sum free by definition.

In 2008, MANFRED DROSTE raised the question whether WANG's result holds for commutative, zero-sum-free semirings. In the present paper, we answer this

[†]The results were achieved in 2008 when the author was employed in MANFRED DROSTE's group at Leipzig University. An extended abstract was presented at *DLT'09* [6].

[‡]Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin, Germany

question positively (see Theorem 3.1(1), below). The proof relies on DICKSON's lemma.

Further, we investigate under which assumptions we can effectively transform a weighted automaton recognizing a series S over a zero-sum free, commutative semiring into an automaton recognizing the support of S. For this, we introduce the zero generation problem (see Sect. 3) and show that the decidability of the zero generation problem is a sufficient and necessary condition for the existence of such an effective transformation. Surprisingly, the computability of the semiring operations is not related to the effectivity of the transformation.

The paper is organized as follows: in Sect. 2, we deal with some preliminaries. In Sect. 3, we present known results and the contribution of the paper. To keep Sect. 3 as a succinct survey, the main proofs are shifted to Sect. 4.

2 Preliminaries

2.1 Notations

Let $\mathbb{N} = \{0, 1, \dots\}$.

Let $n \in \mathbb{N}$. Given a tuple $\bar{x} \in \mathbb{N}^n$, we denote by x_i the i-th component of $\bar{x}$ for $i \in \{1, \dots, n\}$. Given two tuples $\bar{x}, \bar{y} \in \mathbb{N}^n$, we write $\bar{x} \leq \bar{y}$ if $x_i \leq y_i$ for every $i \in \{1, \dots, n\}$. If $\bar{x} \leq \bar{y}$ and $x_i < y_i$ for some $i \in \{1, \dots, n\}$, then we write $\bar{x} < \bar{y}$.

Given a subset $M \subseteq \mathbb{N}^n$, we denote by $\mathsf{Min}(M)$ the set of all minimal tuples of M, that is, $\mathsf{Min}(M) = \{\bar{x} \in M \mid \text{for every } \bar{y} \in M, \bar{y} \leq \bar{x} \text{ implies } \bar{x} = \bar{y}\}$.

The following lemma is well-known in combinatorics, order theory, and commutative algebra. We include its proof for the convenience of the reader.

Lemma 2.1 (DICKSON's lemma). *For every $M \subseteq \mathbb{N}^n$, the set $\mathsf{Min}(M)$ is finite.*

Proof. For $n = 1$, the claim is obvious.

Choose some $n \in \mathbb{N}$, and assume by induction that the claim holds for all subsets of $\mathbb{N}^n$. We show the claim for an arbitrary $M \subseteq \mathbb{N}^{n+1}$.

For $z \in \mathbb{N}$, let

$$M_z := \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n, z) \in \mathsf{Min}(M)\}.$$

Clearly, $\mathsf{Min}(M_z) = M_z$, and hence, M_z is finite by induction. Let

$$M_{\mathbb{N}} := \bigcup_{z \in \mathbb{N}} M_z.$$

By induction $\mathsf{Min}(M_{\mathbb{N}})$ is finite, and thus, there is some $z' \in \mathbb{N}$ such that

$$\mathsf{Min}(M_{\mathbb{N}}) \subseteq \bigcup_{z \leq z'} M_z.$$

Now, we show the claim by showing that

$$\mathsf{Min}(M) \subseteq \bigcup_{z \leq z'} M_z \times \{z\}, \tag{1}$$

i.e., $\mathsf{Min}(M)$ is included in a finite union of finite sets. The notation $M_z \times \{z\}$ means to adjoin z as $(n+1)$-st component to each n-tuple in M_z.

Choose any $\bar{x} \in \mathsf{Min}(M)$. Clearly, $(x_1, \ldots, x_n) \in M_{x_{n+1}} \subseteq M_{\mathbb{N}}$. There is some $\bar{y} \in \mathsf{Min}(M_{\mathbb{N}})$ satisfying $\bar{y} \leq (x_1 \ldots, x_n)$. There is some $z \leq z'$ such that $\bar{y} \in M_z$. If $z < x_{n+1}$ then $(y_1 \ldots, y_n, z) < \bar{x}$ contradicts $\bar{x} \in \mathsf{Min}(M)$. Hence, $x_{n+1} \leq z \leq z'$. Consequently, $\bar{x}$ belongs to the right hand side of (1). □

Let Σ be a finite alphabet. We denote the empty word by ε. We denote by $|w|$ the length of a word $w \in \Sigma^*$. For every $w \in \Sigma^*$, $a \in \Sigma$, let $|w|_a$ be the number of occurrences of the letter a in w.

A monoid $(\mathbb{M}, \cdot, 1)$ consists of a set $\mathbb{M}$ together with a binary associative operation $\cdot$ and an identity 1.

We call a monoid $(\mathbb{M}, \cdot, 1)$ *commutative* if $k\ell = \ell k$ for every $k, \ell \in \mathbb{M}$.

We call $0 \in \mathbb{M}$ a *zero*, if $k0 = 0k = 0$ for every $k \in \mathbb{M}$.

Given a monoid $\mathbb{M}$, $m \in \mathbb{N}$, and $s_1, \ldots, s_m \in \mathbb{M}$, we denote by $\langle s_1, \ldots, s_m \rangle$ the submonoid of $\mathbb{M}$ generated by $s_1, \ldots, s_m$, that is, the smallest monoid $\mathbb{M}' \subseteq \mathbb{M}$ satisfying $s_1, \ldots, s_m \in \mathbb{M}'$.

Given a monoid $\mathbb{M}$, an $s \in \mathbb{M}$, and a submonoid $\mathbb{M}' \subseteq \mathbb{M}$, we denote by $s \cdot \mathbb{M}'$ the set $\{s \cdot s' \mid s' \in \mathbb{M}'\}$.

A *semiring* $(\mathbb{K}, +, \cdot, 0, 1)$ consists of a set $\mathbb{K}$ together with two binary operations $+$ and $\cdot$ such that $(\mathbb{K}, +, 0)$ is a commutative monoid, $(\mathbb{K}, \cdot, 1)$ is a monoid with zero 0, and $(\mathbb{K}, \cdot, 1)$ distributes over $(\mathbb{K}, +, 0)$.

We call a semiring $(\mathbb{K}, +, \cdot, 0, 1)$ *commutative* if $(\mathbb{K}, \cdot, 1)$ is a commutative monoid. We call $\mathbb{K}$ *zero-divisor free* if for every $k, \ell \in \mathbb{K} \setminus \{0\}$, we have $k\ell \neq 0$. We call $\mathbb{K}$ *zero-sum free* if for every $k, \ell \in \mathbb{K} \setminus \{0\}$, we have $k + \ell \neq 0$. Semirings which are both zero-divisor free and zero-sum free are called *positive* semirings.

We call $\mathbb{K}$ *locally finite* if for every finite subset $C \subseteq \mathbb{K}$, there is a finite semiring $\mathbb{K}'$ satisfying $C \subseteq \mathbb{K}' \subseteq \mathbb{K}$.

2.2 Weighted Finite Automata

We recall some notions on (weighted) automata and recommend [1, 2, 4, 7, 8, 10] for overviews.

Let $(\mathbb{K}, +, \cdot, 0, 1)$ be a semiring. Mappings from Σ^* to $\mathbb{K}$ are often called *series*. We denote the class of all series from Σ^* to $\mathbb{K}$ by $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$.

A *weighted finite automaton* (for short *WFA*) over $\mathbb{K}$ is a tuple $[Q, E, \lambda, \varrho]$, where

- Q is a non-empty, finite set of *states*,
- E is a finite subset of $Q \times \Sigma \times \mathbb{K} \times Q$, and
- $\lambda, \varrho : Q \to \mathbb{K}$.

We call the tuples in E *transitions*. For every $q \in Q$, we call $\lambda(q)$ resp. $\varrho(q)$ the *initial weight* resp. *accepting weight* of q. We call states $q \in Q$ which satisfy $\lambda(q) \neq 0$ (resp. $\varrho(q) \neq 0$) *initial* (resp. *accepting*) states.

Let $\mathcal{A} = [Q, E, \lambda, \varrho]$ be a WFA. Let $n \geq 1$. A path π of length n is a sequence

$$(q_0, a_1, s_1, q_1)\,(q_1, a_2, s_2, q_2)\,\ldots\,(q_{n-1}, a_n, s_n, q_n)$$

of transitions in E. We call the word $a_1 \ldots a_n$ the *label* of π. We define $\sigma(\pi) = \lambda(q_0) \cdot s_1 \cdot s_2 \cdot \cdots \cdot s_n \cdot \varrho(q_n)$, the *weight* of π. For every state $q \in Q$, we assume some path from q to q which is labeled with ε and weighted with 1.

For every $p, q \in Q$ and every $w \in \Sigma^*$, we denote by $p \overset{w}{\rightsquigarrow} q$ the set of all paths with label w which start at p and end at q. Then, $\mathcal{A}$ defines a series $|\mathcal{A}| : \Sigma^* \to \mathbb{K}$ by

$$|\mathcal{A}|(w) = \sum_{p,q \in Q,\ \pi \in p \overset{w}{\rightsquigarrow} q} \sigma(\pi)$$

for every $w \in \Sigma^*$.

We call a series $S : \Sigma^* \to \mathbb{K}$ *recognizable* if $S = |\mathcal{A}|$ for some WFA $\mathcal{A}$.

We define the *support* of a series $S : \Sigma^* \to \mathbb{K}$ as

$$\mathsf{supp}(S) = \{w \in \Sigma^* \mid S(w) \neq 0\}.$$

An *(unweighted) automaton* is a tuple $\mathcal{A} = [Q, E, I, F]$, where Q is a finite set, $E \subseteq Q \times \Sigma \times Q$, $I \subseteq Q$, and $F \subseteq Q$.

Let $\mathcal{A} = [Q, E, \lambda, \varrho]$ be an automaton. Let $n \geq 1$. A path π of length n is a sequence

$$(q_0, a_1, q_1)\,(q_1, a_2, q_2)\,\ldots\,(q_{n-1}, a_n, q_n)$$

of transitions in E. As above, we call $a_1 \ldots a_n$ the *label* of π. We call π successful, if $q_0 \in I$ and $q_n \in F$. We denote by $L(\mathcal{A})$ the language of $\mathcal{A}$, that is, the language consisting of all labels of successful paths.

3 Overview, Main Results, and Discussion

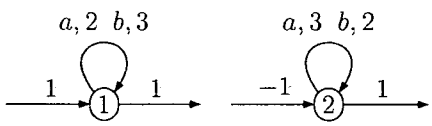
The supports of recognizable series are well-studied objects, see [3, 9] for recent overviews.

It is well known that there are recognizable series S such that $\mathsf{supp}(S)$ is not a recognizable language.

Example 3.1. A folklore example is the series S over the semiring of the integers $(\mathbb{Z}, +, \cdot, 0, 1)$ defined by $S(w) = 2^{|w|_a}3^{|w|_b} - 3^{|w|_a}2^{|w|_b}$. For every $w \in \Sigma^*$, we have $S(w) = 0$ iff $|w|_a = |w|_b$. Hence,

$$\mathsf{supp}(S) = \left\{w \in \Sigma^* \,\middle|\, |w|_a \neq |w|_b\right\}$$

which is not a recognizable language. Nevertheless, S is a recognizable series: just consider the WFA given below.



However, for large classes of semirings, the support of a recognizable series is always a recognizable language. It is well known that these classes include all positive semirings, all finite and moreover even all locally finite semirings [3, 5, 9].

Moreover, WANG [11] defined the notion of a quasi-positive semiring: a semiring $\mathbb{K}$ is called *quasi-positive* if for every $k \in \mathbb{K} \setminus \{0\}$, $\ell \in \mathbb{K}$, $n \in \mathbb{N}$, we have $k^n + \ell \neq 0$. Every positive semiring is quasi-positive, and every quasi-positive semiring is zero-sum free. There are quasi-positive semirings which are not positive. Just let $\mathbb{K} = \mathbb{N} \times \mathbb{N}$ equipped with componentwise addition and multiplication of integers.

Moreover, there are zero-sum free semirings which are not quasi-positive.

Example 3.2. Let $\mathbb{K}$ be the semiring of (2×2)-matrices over the non-negative rational numbers $(\mathbb{Q}_+, +, \cdot, 0, 1)$ and let

$$k = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \ell = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Clearly, $k^2 + \ell$ yields the zero matrix, and hence, $\mathbb{K}$ is not quasi-positive but zero-sum-free.

In the context of our main result, it raises the question for a commutative, zero-sum free semiring which is not quasi-positive. Indeed,[1] let $\mathbb{K}'$ be the subset of $\mathbb{K}$ consisting of all matrices of the form

$$\begin{pmatrix} x & y \\ 0 & x \end{pmatrix} \quad \text{for} \quad x, y \in \mathbb{Q}_+.$$

It is easy to verify that $\mathbb{K}'$ is a commutative subsemiring of $\mathbb{K}$. It is zero-sum-free, and since $k, \ell \in \mathbb{K}'$, it is not quasi-positive.

WANG showed that for every recognizable series S over a commutative, quasi-positive semiring, $\mathsf{supp}(S)$ is recognizable [11]. In 2008, MANFRED DROSTE raised the question whether WANG's result holds for commutative, zero-sum-free semirings in a lecture script on weighted automata theory. In the present paper, we answer this question positively (see Theorem 3.1(1), below). Our approach is quite different from WANG's paper [11], since WANG was mainly interested in other but related questions and achieved his result as a byproduct.

One key observation is that for zero-sum-free semirings, a word w belongs to the support of the series of some WFA iff the WFA admits at least one path for w with a non-zero weight. In contrast to Example 3.1, it cannot happen that the weights of all paths for w are summarized to 0.

Further, we examine under which assumptions we can effectively construct an automaton recognizing $\mathsf{supp}(S)$ from a WFA recognizing S. Surprisingly, the computability of $+$ or $\cdot$ is not related to the effectivity of the construction. To achieve an effective construction, we introduce the *zero generation problem* (for short *ZGP*):

Let $\mathbb{M}$ be a monoid with a zero. An instance of the ZGP consists of two integers $m, m' \in \mathbb{N}$ and $s_1, \ldots, s_m, s'_1, \ldots, s'_{m'} \in \mathbb{M}$. The ZGP means to decide whether

[1] The semiring $\mathbb{K}'$ was provided by an anonymous referee.

$0 \in s_1 \cdots s_m \cdot \langle s'_1, \ldots, s'_{m'} \rangle$, i.e., whether there exists some $s \in \langle s'_1, \ldots, s'_{m'} \rangle$ such that the product $s_1 \cdots s_m \cdot s$ yields zero. The presentation of the ZGP seems to be circumstantial, but we want to avoid using the computability of the product in $\mathbb{M}$.

Note that the integers m and m' in the ZGP are allowed to be 0. Consequently, the problem to decide whether for given $m \in \mathbb{N}$ and $s_1, \ldots, s_m \in \mathbb{M}$, we have $s_1 \cdots s_m = 0$ is a particular case of the ZGP.

We can show that the decidability of the ZGP of the monoid $(\mathbb{K}, \cdot, 1)$ is a sufficient and necessary condition for the effectivity of the construction of the automaton recognizing the support of some recognizable series over a semiring $\mathbb{K}$.

To sum up:

Theorem 3.1. *Let Σ be an alphabet and $(\mathbb{K}, +, \cdot, 0, 1)$ be a zero-sum free, commutative semiring.*

1. *For every recognizable series $S \in \mathbb{K}\langle\langle \Sigma^* \rangle\rangle$, $\mathsf{supp}(S)$ is a recognizable language.*
2. *Assume $|\Sigma| \geq 2$. Given a WFA $\mathcal{A}$ over $\mathbb{K}$, we can effectively construct an automaton which recognizes $\mathsf{supp}(|\mathcal{A}|)$ iff $(\mathbb{K}, \cdot, 1)$ has a decidable ZGP.*

Clearly, the construction in (2) is also effective for $|\Sigma| = 1$. But if $|\Sigma| = 1$ we cannot show that the decidability of the ZGP is a necessary condition.

Unfortunately, we cannot give any reasonable upper bound in the construction in Theorem 3.1(2). Given a WFA $\mathcal{A}$ over a zero-sum free, commutative semiring $\mathbb{K}$, the number of states of an automaton recognizing $\mathsf{supp}(|\mathcal{A}|)$ does not only depend on the number of states of $\mathcal{A}$ and the weights in $\mathcal{A}$, but also it highly depends on structural properties of the semiring $\mathbb{K}$. The construction of the automaton recognizing $\mathsf{supp}(|\mathcal{A}|)$ in the proof of Theorem 3.1(2) involves a certain bound which is computed in a brute search using some algorithm for the ZGP. The existence of this bound is guaranteed by DICKSON's lemma (Lemma 2.1).

4 The Main Proof

4.1 Dickson's Lemma and Computability

Throughout this section, let $(\mathbb{M}, \cdot, 1)$ be a commutative monoid with a zero 0 and let $C = (c_1, \ldots, c_n) \in \mathbb{M}^n$ for some $n \in \mathbb{N}$.

The homomorphism $[\![\,]\!] : (\mathbb{N}^n, +, (0, \ldots, 0)) \to (\mathbb{M}, \cdot, 1)$ defined by

$$[\![\bar{x}]\!] = c_1^{x_1} \cdots c_n^{x_n}$$

for every $\bar{x} = (x_1, \ldots, x_n) \in \mathbb{N}^n$ plays a central role in the entire construction. Let us remark that the commutativity of $\mathbb{M}$ is crucial for the fact that $[\![\,]\!]$ is a homomorphism which will be of crucial importance, e.g., in the proof of Lemma 4.1, below.

We are interested in the set of all $\bar{x} \in \mathbb{N}^n$ satisfying $[\![\bar{x}]\!] = 0$, i.e., we are interested in the set $[\![0]\!]^{-1}$.

Given $\bar{x} \in [\![0]\!]^{-1}$ and $\bar{y} \in \mathbb{N}^n$ satisfying $\bar{x} \leq \bar{y}$, we have $\bar{y} \in [\![0]\!]^{-1}$.

By Lemma 2.1, the set $\mathsf{Min}([\![0]\!]^{-1})$ is finite. We denote by $\mathsf{dg}(C)$ the *degree* of C which is defined as the least non-negative integer such that $\mathsf{Min}([\![0]\!]^{-1})$ is a subset of $\{0,\ldots,\mathsf{dg}(C)\}^n$.

Example 4.1. Let us consider a commutative monoid which admits large degrees. Let $\mathbb{M} := \{q \in \mathbb{Q} \mid 0 \leq q \leq 1\}$. We define an operation $\star$ on $\mathbb{M}$ by setting $p \star q := \mathsf{min}\{p+q, 1\}$ for $p, q \in \mathbb{M}$. Clearly, $(\mathbb{M}, \star, 0)$ is a commutative monoid with zero 1.

Now, let $n \in \mathbb{N}$ and $c_i \in \mathbb{M}$ for $i \in \{1,\ldots,n\}$. If $c_i \neq 0$, then

$$\Big(0,\ldots,0,\underbrace{\left\lceil \tfrac{1}{c_i} \right\rceil}_{i\text{th position}},0,\ldots,0\Big) \quad \in \quad \mathsf{Min}[\![1]\!]^{-1},$$

where $\left\lceil \frac{1}{c_i} \right\rceil$ denotes the least integer larger than or equal to $\frac{1}{c_i}$. Consequently, $\mathsf{dg}(C) \geq \frac{1}{c_i}$.

Given $\bar{x} \in \mathbb{N}^n$ and $z \in \mathbb{N}$, we denote by $\lfloor \bar{x} \rfloor_z$ the tuple defined by $\big(\lfloor \bar{x} \rfloor_z\big)_i = \mathsf{min}\{x_i, z\}$ for every $i \in \{1,\ldots,n\}$.

Lemma 4.1. *For every $\bar{x} \in \mathbb{N}^n$, we have $[\![\bar{x}]\!] = 0$ iff $[\![\lfloor \bar{x} \rfloor_{\mathsf{dg}(C)}]\!] = 0$.*

Proof. We have "$\Leftarrow$", since $\bar{x} \geq \lfloor \bar{x} \rfloor_{\mathsf{dg}(C)}$.

We show "$\Rightarrow$". Since $\bar{x} \in [\![0]\!]^{-1}$, there is a $\bar{y} \in \mathsf{Min}([\![0]\!]^{-1})$ satisfying $\bar{y} \leq \bar{x}$. Let $i \in \{1,\ldots,n\}$. If $x_i \leq \mathsf{dg}(C)$, then $y_i \leq x_i = (\lfloor \bar{x} \rfloor_{\mathsf{dg}(C)})_i$. If $x_i > \mathsf{dg}(C)$, then $y_i \leq \mathsf{dg}(C) = (\lfloor \bar{x} \rfloor_{\mathsf{dg}(C)})_i$ by the definitions of $\mathsf{dg}(C)$ and $\lfloor \bar{x} \rfloor_{\mathsf{dg}(C)}$. Consequently, $\bar{y} \leq \lfloor \bar{x} \rfloor_{\mathsf{dg}(C)}$, and hence, $\lfloor \bar{x} \rfloor_{\mathsf{dg}(C)} \in [\![0]\!]^{-1}$. □

For the effectivity of our construction of the support automaton, it is very important to compute $\mathsf{dg}(C)$ from a given tuple C.

Lemma 4.2. *If the ZGP is decidable in $\mathbb{M}$, then we can effectively compute $\mathsf{dg}(C)$ from C.*

Proof. It suffices to show that for given $n \in \mathbb{N}$, $C = (c_1,\ldots,c_n) \in \mathbb{M}^n$, and $z \in \mathbb{N}$, we can decide whether $z < \mathsf{dg}(C)$. The algorithm can then check for increasing $z \in \{0,1,2,\ldots\}$ whether $z < \mathsf{dg}(C)$, and put out the least z which does not satisfy $z < \mathsf{dg}(C)$.

So assume n, C, z as above. We want to show that $z < \mathsf{dg}(C)$ iff there exists a tuple $\bar{x} \in \{0,\ldots,z\}^n$ which satisfies the following properties:

1. We have $x_i = z$ for some $i \in \{1,\ldots,n\}$.
2. We have $[\![\bar{x}]\!] \neq 0$. Given C and $\bar{x}$, it is decidable whether $[\![\bar{x}]\!] \neq 0$ by the decidability of the ZGP.
3. There is some $\bar{y} \in \mathbb{N}^n$ such that $\bar{x} = \lfloor \bar{y} \rfloor_z$ and $[\![\bar{y}]\!] = 0$.

Given C and $\bar{x}$, this condition is decidable as follows: Let $m = \sum_{i=1}^{n} x_i$. Let $s_1, \ldots, s_m$ be the list over $\mathbb{M}$ constructed by putting x_1 times c_1, x_2 times c_2, ..., and x_n times c_n. We have $s_1 \cdots s_m = [\![\bar{x}]\!]$.

Let $m' \geq 1$ and $s'_1, \ldots, s'_{m'} \in \mathbb{M}$ be a list of the c_i's for the $i \in \{1, \ldots, n\}$ satisfying $x_i = z$.

Clearly, there exists some $\bar{y} \in \mathbb{N}^n$ such that $\bar{x} = \lfloor \bar{y} \rfloor_z$ and $[\![\bar{y}]\!] = 0$ iff $0 \in s_1 \cdots s_m \cdot \langle s'_1, \ldots, s'_{m'} \rangle$. The latter condition is decidable.

Assume $z < \mathsf{dg}(C)$. Choose a $\bar{y} \in \mathsf{Min}([\![0]\!]^{-1})$ such that at least one entry of $\bar{y}$ equals $\mathsf{dg}(C)$. Let $\bar{x} = \lfloor \bar{y} \rfloor_z$. Obviously, $\bar{x}$ satisfies (1) and (3). Since $\bar{x} < \bar{y}$, we have $\bar{x} \notin [\![0]\!]^{-1}$, and hence, $\bar{x}$ satisfies (2).

Assume $z \geq \mathsf{dg}(C)$. Let $\bar{x}, \bar{y} \in \mathbb{N}^n$ such that (1) and (3) are satisfied. From Lemma 4.1, it follows $[\![\lfloor \bar{y} \rfloor_{\mathsf{dg}(C)}]\!] = 0$. Since $\mathsf{dg}(C) \leq z$, we have $\lfloor \bar{y} \rfloor_{\mathsf{dg}(C)} \leq \lfloor \bar{y} \rfloor_z = \bar{x}$, and hence, $[\![\bar{x}]\!] = 0$, i.e., $\bar{x}$ does not satisfy (2).

An algorithm to decide whether $z < \mathsf{dg}(C)$ can check by brute force whether there is an $\bar{x} \in \{0, \ldots, z\}^n$ which satisfies (1), (2), and (3). □

4.2 The Construction of a Support Automaton

Proof of Theorem 3.1. In the first part of the proof we prove (1) and "⇐" in (2).

Let S be the series computed by a WFA $\mathcal{A} = [Q, E, \lambda, \varrho]$.

Let C be a sequence (without repetition) of all weights occurring in $\mathcal{A}$. That is, let $n \in \mathbb{N}$ and $C = (c_1, \ldots, c_n) \in \mathbb{K}^n$ such that:

- For every $i \in \{1, \ldots, n\}$, there is a transition $(p, a, c_i, q) \in E$ or there is a $q \in Q$ satisfying $\lambda(q) = c_i$ or $\varrho(q) = c_i$.
- For every $(p, a, s, q) \in E$, there is exactly one $i \in \{1, \ldots, n\}$ satisfying $c_i = s$.
- For every $q \in Q$, there is exactly one $i \in \{1, \ldots, n\}$ satisfying $\lambda(q) = c_i$, and there is exactly one $i \in \{1, \ldots, n\}$ satisfying $\varrho(q) = c_i$.

We construct an (unweighted) automaton $\mathcal{A}_s$. We will use $\mathsf{dg}(C)$ in a crucial way. If the ZGP is decidable, we can effectively compute $\mathsf{dg}(C)$ by Lemma 4.2 and then, our construction is effective.

The state set of $\mathcal{A}_s$ is $Q_s = \{0, \ldots, \mathsf{dg}(C)\}^n \times Q$.

A state $(\bar{x}, q) \in Q_s$ is an initial state iff there exists some $i \in \{1, \ldots, n\}$ such that

- $x_i = 1$, $\lambda(q) = c_i$, and
- for every $j \in \{1, \ldots, n\}$, $j \neq i$, we have $x_j = 0$.

Consequently, $[\![\bar{x}]\!] = c_i = \lambda(q)$. We denote the set of initial states by I_s.

We could also define the set of initial states by $I'_s = \{(\bar{x}, q) \in Q_s \mid [\![\bar{x}]\!] = \lambda(q)\}$ which is a superset of I_s. One can easily construct examples for which $I_s \subsetneq I'_s$. Just consider the case that for some $(\bar{x}, q) \in Q_s$, we have $x_1 = x_2 = 1$, $x_3 = \cdots = x_n = 0$

and $c_1c_2 = \lambda(q)$. Our construction below remains correct even if we use I'_s instead of I_s. However, the definition of I'_s involves the decision problem $[\![\bar{x}]\!] = \lambda(q)$ which we want to avoid to get an effective construction.

We define a partial mapping $\oplus : \{0,\dots,\mathsf{dg}(C)\}^n \times \mathbb{K} \dashrightarrow \{0,\dots,\mathsf{dg}(C)\}^n$. The key idea behind $\oplus$ is that given $m \in \mathbb{N}$, $s_1,\dots,s_m \in \mathbb{K}$, the operation

$$(\cdots((\bar{x} \oplus s_1) \oplus s_2)\cdots \oplus s_m)$$

counts (up to $\mathsf{dg}(C)$) the number of occurrences of the c_i's in the sequence $s_1,\dots,s_m$.

Let $\bar{x} \in \{0,\dots,\mathsf{dg}(C)\}^n$ and $s \in \mathbb{K}$. We define $\bar{x} \oplus s$ iff there is some $i \in \{1,\dots,n\}$ satisfying $c_i = s$. Let $\bar{y} \in \{0,\dots,\mathsf{dg}(C)\}^n$ be defined by

$$y_j = \begin{cases} x_j + 1 & \text{if } j = i \\ x_j & \text{if } j \neq i. \end{cases}$$

We define $\bar{x} \oplus s = \lfloor \bar{y} \rfloor_{\mathsf{dg}(C)}$.

A state $(\bar{x}, q) \in Q_s$ is an accepting state iff $[\![\bar{x} \oplus \varrho(q)]\!] \neq 0$. Using the decidability of the ZGP, we can decide whether $(\bar{x}, q)$ is an accepting state. We denote the set of accepting states by F_s.

Let $(\bar{x}, p), (\bar{y}, q) \in Q_s$ and $a \in \Sigma$. The triple $\big((\bar{x}, p), a, (\bar{y}, q)\big)$ is a transition in E_s iff there exists a transition $(p, a, s, q) \in E$ satisfying $\bar{x} \oplus s = \bar{y}$. We say that $\big((\bar{x}, p), a, (\bar{y}, q)\big)$ stems from $(p, a, s, q) \in E$.

Let $\mathcal{A}_s = [Q_s, E_s, I_s, F_s]$. We want to show $L(\mathcal{A}_s) = \mathsf{supp}(S)$.

Let $w \in L(\mathcal{A}_s)$. There are $(\bar{x}_0, q_0) \in I_s$, $(\bar{x}_{|w|}, q_{|w|}) \in F_s$, and some path $\pi \in (\bar{x}_0, q_0) \overset{w}{\rightsquigarrow} (\bar{x}_{|w|}, q_{|w|})$ satisfying $[\![\bar{x}_{|w|} \oplus \varrho(q_{|w|})]\!] \neq 0$.

We denote the states of π by $(\bar{x}_0, q_0), (\bar{x}_1, q_1), \dots, (\bar{x}_{|w|}, q_{|w|})$.

For $j \in \{1,\dots,|w|\}$, let $t_j \in E$ such that the j-th transition of π stems from t_j. Clearly, $t_1 \cdots t_{|w|} \in q_0 \overset{w}{\rightsquigarrow} q_{|w|}$ is a path in $\mathcal{A}$.

For every $j \in \{1,\dots,|w|\}$, let $s_j \in \mathbb{K}$ be the weight of t_j. For $j \in \{0,\dots,|w|\}$, let $\bar{y}_j \in \mathbb{N}^n$ be the tuple such that for every $i \in \{1,\dots,n\}$, $y_{j,i}$ is the number of occurrences of c_i among $\lambda(q_0), s_1, \dots, s_j$. In particular $\bar{y}_0 = \bar{x}_0$.

Let $\bar{y} \in \mathbb{N}^n$ such that for every $i \in \{1,\dots,n\}$, y_i is the number of occurrences of c_i among $\lambda(q_0), s_1, \dots, s_{|w|}, \varrho(q_{|w|})$. Clearly, $[\![\bar{y}]\!]$ is the weight of the path $t_1 \cdots t_{|w|}$.

By a straightforward inductive argument, we can show that for every $j \in \{0,\dots,|w|\}$, $\bar{x}_j = \lfloor \bar{y}_j \rfloor_{\mathsf{dg}(C)}$, and $\bar{x}_{|w|} \oplus \varrho(q_{|w|}) = \lfloor \bar{y} \rfloor_{\mathsf{dg}(C)}$.

Since $(\bar{x}_{|w|}, q_{|w|}) \in F_s$, we have $[\![\bar{x}_{|w|} \oplus \varrho(q_{|w|})]\!] \neq 0$, and hence, $[\![\lfloor \bar{y} \rfloor_{\mathsf{dg}(C)}]\!] \neq 0$. By Lemma 4.1, we have $[\![\bar{y}]\!] \neq 0$, i.e., the weight of the path $t_1 \cdots t_{|w|}$ is different from 0. Since $\mathbb{K}$ is zero-sum-free, we have $w \in \mathsf{supp}(|\mathcal{A}|)$.

Thus, we have shown $L(\mathcal{A}_s) \subseteq \mathsf{supp}(|\mathcal{A}|)$. To show $L(\mathcal{A}_s) \supseteq \mathsf{supp}(|\mathcal{A}|)$, we can proceed in the same way. We assume some $w \in \mathsf{supp}(|\mathcal{A}|)$, some accepting path $t_1 \dots t_{|w|}$ with non-zero weight for w in $\mathcal{A}$. For $j \in \{1,\dots,|w|\}$, we denote $t_j = (q_{j-1}, a_j, s_j, q_j)$. Let $\bar{x}_0 = (0,\dots,0) \oplus \lambda(q_0)$. For $j \in \{1,\dots,|w|\}$, let $\bar{x}_j = \bar{x}_{j-1} \oplus s_j$. We can argue as above to show that the transitions $\big((\bar{x}_{i-1}, q_{i-1}), a_j, (\bar{x}_i, q_i)\big)$ form an accepting path for w in $\mathcal{A}_s$. To sum up, $L(\mathcal{A}_s) = \mathsf{supp}(|\mathcal{A}|)$.

We have shown (1) and "$\Leftarrow$" in (2). It remains to show "$\Rightarrow$" in (2). Assume an instance of the ZGP, i.e., let $m, m' \in \mathbb{N}$ and $s_1, \ldots, s_m, s'_1, \ldots, s'_{m'} \in \mathbb{K}$.

Let $w_1, \ldots, w_{m'} \in \Sigma^*$ be mutually distinct, non-empty words of equal length.[2]

We sketch the construction of a WFA $\mathcal{A}$. It has just one initial and one accepting state. The initial and accepting weights are 1. Let a be some letter from Σ. For now, there is exactly one path from the initial to the accepting state. This path is labeled with a^m. The transition weights along this path are $s_1, \ldots, s_m$. For every $j \in \{1, \ldots, m'\}$, we add a loop at the accepting state which is labeled with w_j. The first transition of the loop is weighted with s'_j, the remaining transitions of the loop are weighted with 1.

For every n and $i_1, \ldots, i_n \in \{1, \ldots, m'\}$, we have

$$|\mathcal{A}|(a^m w_{i_1} \ldots w_{i_n}) = s_1 \cdots s_m \cdot s'_{i_1} \cdots s'_{i_n}.$$

Moreover, we have $\mathsf{supp}(|\mathcal{A}|) = a^m\{w_1, \ldots, w_{m'}\}^*$ iff $0 \notin s_1 \cdots s_m \cdot \langle s'_1 \cdots s'_{m'} \rangle$.

By the assumption of "$\Rightarrow$" in (2), we can effectively construct an automaton $\mathcal{A}_s$ which recognizes $\mathsf{supp}(|\mathcal{A}|)$. By checking $L(\mathcal{A}_s) = a^m\{w_1, \ldots, w_{m'}\}^*$, we can check whether $\mathsf{supp}(|\mathcal{A}|) = a^m\{w_1, \ldots, w_{m'}\}^*$, i.e., whether $0 \notin s_1 \cdots s_m \cdot \langle s'_1 \cdots s'_{m'} \rangle$. □

Acknowledgements

The author thanks the anonymous reviewers of the present paper and its extended abstract at DLT'09 [6]. The author greatly acknowledges the example of a commutative, zero-sum free semiring which is not quasi-positive provided by an anonymous referee shown in Example 3.2.

References

[1] Berstel, J. *Transductions and Context-Free Languages.* B. G. Teubner, Stuttgart, 1979.

[2] Berstel, J. and Reutenauer, C. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Berlin Heidelberg New York, 1984.

[3] Berstel, J. and Reutenauer, C. *Noncommutative Rational Series With Applications*, volume 137 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 2010.

[4] Droste, M., Kuich, W., and Vogler, H., editors. *Handbook of Weighted Automata.* Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2009.

[2] At this point, we need $|\Sigma| > 1$.

[5] Kirsten, D. An algebraic characterization of semirings for which the support of every recognizable series is recognizable. In Královič, R. and Niwiński, D., editors, *MFCS'09 Proceedings*, volume 5734 of *LNCS*, pages 489–500. Springer-Verlag, Berlin, 2009.

[6] Kirsten, D. The support of a recognizable series over a zero-sum free, commutative semiring is recognizable. In Diekert, V. and Nowotka, D., editors, *DLT'09 Proceedings*, volume 5583 of *LNCS*, pages 326–333. Springer-Verlag, Berlin, 2009.

[7] Kuich, W. Semirings and formal power series. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages, Vol. 1, Word, Language, Grammar*, pages 609–677. Springer-Verlag, Berlin, 1997.

[8] Reutenauer, C. A survey on noncommutative rational series. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 24:159–169, 1996.

[9] Sakarovitch, J. Rational and recognisable power series. Chapter 4 in [4], 2009.

[10] Salomaa, A. and Soittola, M. *Automata-Theoretic Aspects of Formal Power Series.* Texts and Monographs on Computer Science. Springer-Verlag, Berlin Heidelberg New York, 1978.

[11] Wang, H. On rational series and rational languages. *Theoretical Computer Science*, 205(1-2):329–336, 1998.

Acta Cybernetica **20** (2011) 223–250.

Survey: Weighted Extended Top-down Tree Transducers Part I — Basics and Expressive Power

Andreas Maletti*

Abstract

Weighted extended top-down tree transducers (transducteurs généralisés descendants [Arnold, Dauchet: *Bi-transductions de forêts.* ICALP'76. Edinburgh University Press. 1976]) received renewed interest in the field of Natural Language Processing, where they are used in syntax-based machine translation. This survey presents the foundations for a theoretical analysis of weighted extended top-down tree transducers. In particular, it discusses essentially complete semirings, which are a novel concept that can be used to lift incomparability results from the unweighted case to the weighted case even in the presence of infinite sums. In addition, several equivalent ways to define weighted extended top-down tree transducers are presented and the individual benefits of each presentation is shown on a small result.

Keywords: tree transducer, weighted tree transducer, expressive power

1 Introduction

Tree automata theory [24, 25] and computational linguistics [42, 30] were tightly intertwined at their inception. In particular, top-down tree transducers were devised by THATCHER [47] and ROUNDS [44] for applications in natural language processing (NLP). However, this tight connection was lost early on and the two fields went separate ways. NLP research focussed on the algorithmic and scaling issues, whereas the tree automata theory research focussed on refining and extending models of automata and transducers. In particular, the following devices were investigated:

- bottom-up tree transducers [48] and attributed tree transducers [18],
- macro tree transducers [7, 15] and modular tree transducers [16],

*Universitat Rovira i Virgili, Departament de Filologies Romàniques, Avinguda Catalunya 35, 43002 Tarragona, Spain. E-mail: andreas.maletti@urv.cat. The author now is at: Universität Stuttgart, Institut for Natural Language Processing, Azenbergstraße 12, 70174 Stuttgart, Germany

- tree bimorphisms [3], and various models with synchronization (e.g., [43]).

Due to the technical difficulties and algorithmic and scaling complexities encountered with tree automata, computational linguists had reverted to finite-state string transducers [29], which are simple to understand, easy to train even on large amounts of data, and have nice theoretical properties. However, finite-state string transducers are not expressive enough for many applications in natural language processing [32]. This realization recently sparked a revival of tree automata in NLP research.

SHIEBER [46] and others have argued that the classical top-down tree transducers [47, 44] are generally inadequate for linguistic tasks without the use of copying and deletion. In general, copying causes many operations to become intractable or impossible, which severely limits even the use of copying top-down tree transducers.

A promising alternative is the *extended top-down tree transducer*, which was originally conceived by [44] and has been further pursued by [8, 2, 27, 33, 41]. In this survey we provide an in-depth review of some of the results for weighted extended top-down tree transducers. In fact, we assume that the reader has some fundamental understanding of unweighted tree automata theory. The goal is to present the common definitions and provide an overview of the various techniques used in the weighted setting. In this aspect this survey differs significantly from [41], which provides a detailed explanation of unweighted extended top-down tree transducers, their expressive power, and their essential features. In addition, [23] surveys results on weighted top-down tree transducers. Contrary to traditional research papers, we do not aim for the most general results here, but rather try to keep the material accessible.

In this survey, we introduce the theoretical foundations and showcase a general method that allows to lift inequalities from the unweighted case to the weighted setting. This is prepared in the first section with a detailed review of semirings and some required properties. Then we introduce the basic notions and notations for handling trees before we finally introduce the main model, the weighted extended top-down tree transducer, in Section 3. Our reference semantics is based on rewriting, but we provide an alternative semantics that corresponds to the initial-algebra semantics [23] of weighted top-down tree transducers. In addition, we relate the linear and nondeleting version of weighted extended top-down tree transducers to linear and complete bimorphisms [3]. The benefits of those additional representations and semantics are illustrated on typical constructions in Section 4. Moreover, in Section 4 we demonstrate a general method to lift known results from the unweighted setting to the weighted setting. We use a part of the HASSE diagram of [41] for classes of tree transformations computed by unweighted extended top-down tree transducers and show how to translate it to the weighted setting using the semiring basics introduced in Section 2.

Overall, we provide proof details whenever instructive, but it is generally safe to skip them. However, the main purpose of this survey is to showcase the techniques, so the proofs generally contain important information. Moreover, we provide examples for some interesting concepts and constructions. We refer to the original research papers for the details and the most general forms of the statements re-

produced or reported here. In addition, we refer to [23] for a survey of weighted top-down tree transducers and to the forth-coming paper [21], which discusses an even more general weighted model and contains some of the results reported here.

2 The Basics

In this section, we first recall the definition and cover some basic properties of our weight structures: semirings [28, 26]. Recently, more general weight structures such as bi-monoids have been proposed [10] for weighted automata and transducers on strings and trees, but we will focus exclusively on semirings in this survey. To keep the presentation self-contained, we try to present proof details whenever possible.

2.1 Semirings

Let $\cdot\colon A^2 \to A$ be a binary operation on a set A, which we write using juxtaposition (i.e., ab stands for $a \cdot b$). It is *associative* if $(ab)c = a(bc)$ for every $a, b, c \in A$. Moreover, the operation $\cdot$ is *commutative* if $ab = ba$ for every $a, b \in A$. An element $1 \in A$ is a *neutral element* if $1a = a = a1$ for every $a \in A$. Finally, an element $0 \in A$ is *absorbing* if $a0 = 0 = 0a$ for every $a \in A$. In general, operations that use "multiplicative" operation signs (like $\cdot$, $\times$, $\otimes$, $\prod$, etc.) have precedence over operations written using "additive" signs (like $+$, $\oplus$, $\sum$, etc.). Thus, $ab + c$ stands for $(a \cdot b) + c$ and $\sum_{i \in I} a_i b_i$ stands for $\sum_{i \in I}(a_i \cdot b_i)$. As with $\cdot$ we often drop multiplicative symbols altogether and write a product $a \otimes b$ simply as the juxtaposition ab. As usual, the product $a \cdots a$ containing $n \in \mathbb{N}$ factors $a \in A$ is abbreviated by a^n.

A *semiring* [28, 26] is an algebraic structure $(A, +, \cdot, 0, 1)$ with two binary operations $+, \cdot\colon A^2 \to A$ and two constants $0, 1 \in A$ such that

- $+$ and $\cdot$ are associative operations, of which $+$ is also commutative,
- $\cdot$ distributes over $+$ from both sides, which means that $(a+b)c = ac+bc$ and $a(b+c) = ab+ac$ for every $a, b, c \in A$,
- 0 and 1 are the neutral elements for $+$ and $\cdot$, respectively, and
- 0 is absorbing for $\cdot$.

In other words, the semiring $(A, +, \cdot, 0, 1)$ consists of the commutative (additive) monoid $(A, +, 0)$ and the (multiplicative) monoid $(A, \cdot, 1)$ such that $\cdot$ distributes (both-sided) over finite sums $\sum_{i=1}^{k} a_i$. The absorption property

$$a \cdot 0 = a \cdot \sum_{i=1}^{0} a_i = \sum_{i=1}^{0} a a_i = 0$$

corresponds to distributivity over the empty sum $\sum_{i=1}^{0} a_i = 0$. If $\cdot$ is also commutative, then the semiring is *commutative*. It is a *ring* if there exists an element $-1 \in A$ such that $(-1) + 1 = 0$. By distributivity this yields that in a ring every element $a \in A$ has an *additive inverse* $-a \in A$. A ring is a *field* if for every $a \in A$ there exists a *multiplicative inverse* $a^{-1} \in A$ such that $aa^{-1} = 1 = a^{-1}a$. Finally, given two

semirings $(A, +, \cdot, 0, 1)$ and $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ and a mapping $h\colon A \to S$, the mapping h is a *semiring homomorphism* if $h(0) = \mathbf{0}$, $h(1) = \mathbf{1}$, $h(a + b) = h(a) \oplus h(b)$, and $h(a \cdot b) = h(a) \odot h(b)$ for every $a, b \in A$. Consequently, a semiring homomorphism is compatible with finite sums and products. For every mapping $f\colon B \to A$, we let $\operatorname{supp}(f) = \{b \in B \mid f(b) \neq 0\}$.

Commonly used semirings include

- the commutative BOOLEAN semiring $(\{0, 1\}, \max, \min, 0, 1)$ where 0 can be understood as *false* and 1 as *true*,
- the commutative semiring of natural numbers $(\mathbb{N}, +, \cdot, 0, 1)$ with the usual addition and multiplication,
- the tropical semiring $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$, which is also commutative,
- the commutative field $(\mathbb{R}, +, \cdot, 0, 1)$ of real numbers, and
- the (typically non-commutative) semiring $(A^{Q \times Q}, \mathbin{\underline{+}}, \mathbin{\underline{\cdot}}, \underline{0}, \underline{1})$ of $Q \times Q$-matrices over a semiring $(A, +, \cdot, 0, 1)$ for every finite set Q where $\mathbin{\underline{+}}$ and $\mathbin{\underline{\cdot}}$ are the usual operations of matrix addition and multiplication, respectively, $\underline{0}$ is the zero matrix, and $\underline{1}$ is the unit matrix.

A semiring $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ is a *subsemiring* of a semiring $(A, +, \cdot, 0, 1)$ if $S \subseteq A$ and the identity mapping $\mathrm{id}\colon S \to A$ such that $\mathrm{id}(s) = s$ for every $s \in S$ is a semiring homomorphism. In other words, $\mathbf{0} = 0$, $\mathbf{1} = 1$, $a \oplus b = a + b$, and $a \odot b = ab$ for every $a, b \in S$, which states that the subsemiring shares the constants 0 and 1 and uses the same operations restricted to its (smaller) carrier set. Given a set $S \subseteq A$, we denote by $\langle S \rangle$ the carrier set of the subsemiring *generated by* S; i.e., the carrier set of the smallest subsemiring of $(A, +, \cdot, 0, 1)$ that contains S. The subsemiring $(S, \oplus, \odot, 0, 1)$ is *finitely generated* if there exists a finite set $C \subseteq A$ such that $S = \langle C \rangle$. Due to the distributivity law, every element $a \in \langle S \rangle$ with $S \subseteq A$ can be presented as $\sum_{i=1}^{k} \bigl(\prod_{j=1}^{n_i} a_{ij}\bigr)$ with $k, n_1, \ldots, n_k \in \mathbb{N}$ and $a_{ij} \in S$ for every $1 \leq i \leq k$ and $1 \leq j \leq n_i$. This representation as a sum of products will be important later on.

Next, we discuss infinite sums in a semiring $(A, +, \cdot, 0, 1)$. Since we will only encounter sums with countably many summands, we only define *countably complete semirings* [11, 36, 28, 26, 31]. Recall that a set I is *countable* if $|I| \leq \aleph_0$ where $\aleph_0 = |\mathbb{N}|$ (i.e., if it has at most as many elements as the natural numbers). Moreover, a *partition* of a set I is a set $\mathcal{J}$ of nonempty, pairwise disjoint subsets of I such that their union is I. Formally, $\mathcal{J}$ is a partition of I if (i) $\emptyset \notin \mathcal{J}$, (ii) $J \cap J' = \emptyset$ for every $J, J' \in \mathcal{J}$ with $J \neq J'$, and (iii) $I = \bigcup_{J \in \mathcal{J}} J$. Note that any partition of a countable set is itself countable. An infinitary sum operation $\sum$ is a family $(\sum_I)_I$ of mappings $\sum_I\colon A^I \to A$ for every countable index set I. We generally write $\sum_{i \in I} f(i)$ instead of $\sum_I f$. The semiring $(A, +, \cdot, 0, 1)$ together with the infinitary sum operation $\sum$ is *countably complete* if

(B) $\sum_{i \in \{j, j'\}} a_i = a_j + a_{j'}$ for all $j \neq j'$ and $a_j, a_{j'} \in A$,

(P) $\sum_{i \in I} a_i = \sum_{J \in \mathcal{J}} \bigl(\sum_{i \in J} a_i\bigr)$ for every countable index set I, partition $\mathcal{J}$ of I, and element $a_i \in A$ with $i \in I$, and

(D) $a\bigl(\sum_{i \in I} a_i\bigr) = \sum_{i \in I} a a_i$ and $\bigl(\sum_{i \in I} a_i\bigr)a = \sum_{i \in I} a_i a$ for every $a \in A$, countable index set I, and $a_i \in A$ with $i \in I$.

The axioms (B), (P), and (D) are also called binary sum, partition, and distributivity axiom, respectively, and they guarantee that the usual laws of associativity, commutativity and distributivity also hold for sums of countably many summands. The literature often also lists the following two additional axioms (E) $\sum_{i\in\emptyset} a_i = 0$ and (U) $\sum_{i\in\{j\}} a_i = a_j$. Next we show that they are valid in any countably complete semiring, and we will use them freely after the proof. In addition, we present another interesting known property [28, 26] of such semirings.

Proposition 1. *If* $(A, +, \cdot, 0, 1)$ *is countably complete with respect to* $\sum$*, then*

- $\sum_{i\in I} 0 = 0$ *for every countable index set* I*,*
- $\sum_{i\in\{j\}} a_i = a_j$ *for every* j *and* $a_j \in A$*, and*
- $a + b = 0$ *implies* $a = 0 = b$ *for every* $a, b \in A$*.*

Proof. To illustrate the handling of infinite sums, we prove these statements formally. For the third item, we present the proof of [26, Proposition 22.28]. First, the distributivity axiom immediately yields

$$\sum_{i\in I} 0 = \sum_{i\in I} 0a_i \overset{\text{(D)}}{=} 0 \cdot \sum_{i\in I} a_i = 0$$

for every countable index set I and $a_i \in A$ with $i \in I$. This proves the first item and Axiom (E), which is a special case of the first item. Next, let j be an arbitrary element and $a_j \in A$. We prove the unary sum axiom (U) by adding the neutral element 0 to a. In this way, we can form binary sums. Let j' be such that $j' \neq j$ and $a_{j'} = 0$. Then

$$\sum_{i\in\{j\}} a_i = \Big(\sum_{i\in\{j\}} a_i\Big) + 0 \overset{\dagger}{=} \Big(\sum_{i\in\{j\}} a_i\Big) + \Big(\sum_{i\in\{j'\}} a_i\Big) \overset{\text{(B)}}{=} \sum_{J\in\{\{j\},\{j'\}\}} \Big(\sum_{i\in J} a_i\Big)$$
$$\overset{\text{(P)}}{=} \sum_{i\in\{j,j'\}} a_i \overset{\text{(B)}}{=} a_j + a_{j'} = a_j + 0 = a_j$$

where we used the first item in the step marked †. For the last statement, let $a, b \in A$ be such that $a + b = 0$. Consider the following two partitions of $\mathbb{N}$:

$$\mathcal{J} = \{\{2j, 2j+1\} \mid j \in \mathbb{N}\}$$
$$\mathcal{J}' = \{\{0\}\} \cup \{\{2j+1, 2j+2\} \mid j \in \mathbb{N}\} \ .$$

Moreover, for every $i \in \mathbb{N}$, let $a_i = a$ if i is even and $a_i = b$ otherwise. Intuitively, consider the sum $\sum_{i\in\mathbb{N}} a_i$. The partition $\mathcal{J}$ always pairs a_{2j} and a_{2j+1}, which are a and b, respectively. Thus, all subsums under this partition will be $a + b = 0$. The partition $\mathcal{J}'$ is similar. It pairs a_{2j+1} and a_{2j+2}, which are b and a, respectively, but it keeps $a_0 = a$ separate. Thus, the subsum for $\{0\}$ will be a and the remaining subsums will be $b + a = 0$. Using the first item, we can then prove that $\sum_{i\in\mathbb{N}} a_i = 0$ using the partition $\mathcal{J}$ because all the subsums are 0, but we can also prove that $\sum_{i\in\mathbb{N}} a_i = a$ using the partition $\mathcal{J}'$. Clearly, this proves that $a = 0$. Formally, let

$\mathcal{J}'' = \mathcal{J}' \setminus \{\{0\}\}$. Then

$$a = a + 0 \overset{(U),\dagger}{=} \Big(\sum_{i\in\{0\}} a_i\Big) + \sum_{J\in\mathcal{J}''} 0 = \Big(\sum_{i\in\{0\}} a_i\Big) + \sum_{J\in\mathcal{J}''} (b + a)$$
$$\overset{(B)}{=} \Big(\sum_{i\in\{0\}} a_i\Big) + \sum_{J\in\mathcal{J}''} \Big(\sum_{i\in J} a_i\Big) \overset{(P)}{=} \Big(\sum_{i\in\{0\}} a_i\Big) + \sum_{i\in\mathbb{N}_+} a_i \overset{(B)}{=} \sum_{J\in\{\{0\},\mathbb{N}_+\}} \Big(\sum_{i\in J} a_i\Big)$$
$$\overset{(P)}{=} \sum_{i\in\mathbb{N}} a_i \overset{(P)}{=} \sum_{J\in\mathcal{J}} \Big(\sum_{i\in J} a_i\Big) \overset{(B)}{=} \sum_{J\in\mathcal{J}} (a + b) = \sum_{J\in\mathcal{J}} 0 \overset{\dagger}{=} 0 \ ,$$

where we used the first item in the steps marked †. Clearly, this also yields that $b = 0 + b = a + b = 0$. □

Now let Q be a finite set and $(A, +, \cdot, 0, 1)$ be a countably complete semiring with respect to $\sum$. Then the matrix semiring $(A^{Q\times Q}, \dot{+}, \dot{\cdot}, \underline{0}, \underline{1})$ is a (typically non-commutative) semiring that is countably complete with respect to the usual generalization of matrix addition to countable sums. Consequently, we can define a $Q \times Q$-matrix M^* for every $Q \times Q$-matrix $M \in A^{Q\times Q}$ over A by $M^* = \sum_{n\in\mathbb{N}} M^n$. Recall that $M^0 = \underline{1}$ and $M^{n+1} = M \dot{\cdot} M^n$ for every $n \in \mathbb{N}$. If we interpret M as the incidence matrix of a weighted graph, then the entry $M^*_{p,q}$ with $p, q \in Q$ equals the sum of the weights of all paths leading from p to q where the weight of a path is obtained by multiplying the weights of the edges. For example, in the tropical semiring, $M^*_{p,q}$ equals the smallest weight of path from p to q.

Proposition 2. *Every ring $(A, +, \cdot, 0, 1)$ with $0 \neq 1$ is not countably complete with respect to any $\sum$.*

Proof. Suppose that it is countably complete with respect to some $\sum$. Since there exists an element $-1 \in A$ such that $(-1) + 1 = 0$, we conclude by Proposition 1 that $-1 = 0 = 1$, which contradicts the assumption $0 \neq 1$. □

Finally, we consider semiring homomorphisms that preserve certain countably infinite sums. Let $(A, +, \cdot, 0, 1)$ be a countably complete semiring with respect to $\sum$ and $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ be a countably complete semiring with respect to $\bigoplus$. In addition, let $h\colon A \to S$ be a semiring homomorphism and $B \subseteq A$. Then h is *B-complete* if $h(\sum_{i\in I} a_i) = \bigoplus_{i\in I} h(a_i)$ for every countable index set I and every $a_i \in B$ with $i \in I$. A semiring homomorphism is *essentially complete* [21] if it is $\langle B\rangle$-complete for every finite set $B \subseteq A$. The traditional notion of a *complete* semiring homomorphism [34, 17] requires that it is A-complete. In other words, an essentially complete homomorphism preserves countable sums, of which the summands all belong to a finitely-generated subsemiring, whereas the traditional notion requires that all countable sums need to be preserved. The relaxed notion of 'essential completeness' is typically sufficient for weighted finite-state devices because their finitely many transitions carry only finitely many weights. Thus, most weighted finite-state devices [36, 9] (whether over trees, strings, pictures, etc.) compute in a finitely-generated subsemiring.

We say that a semiring is *proper* if it is not a ring. For example, the BOOLEAN semiring is proper. WANG proved in [49, Theorem 2.1] and [50, Lemma 3.1] that for every proper commutative semiring there exists a semiring homomorphism from it to the BOOLEAN semiring. This important result essentially yields that weighted devices over proper commutative semirings behave as the corresponding unweighted (i.e., BOOLEAN weighted) devices. Here, we extend this result to include infinite summation, which is present in some finite-state models [9]. However, we first recall the original construction of the semiring homomorphism by [49, 50]. To this end, we introduce some additional notions for a commutative semiring $(A, +, \cdot, 0, 1)$. A set $C \subseteq A$ is a *co-ideal* if

- $cc' \in C$ for all $c, c' \in C$ and
- $a + c \in C$ for every $a \in A$ and $c \in C$.

In other words, co-ideals are closed under multiplication of its elements and closed under addition of one of its elements with any semiring element. Dually, an *ideal* $I \subseteq A$ is such that

- $ai \in I$ for every $a \in A$ and $i \in I$ and
- $i + i' \in I$ for all $i, i' \in I$.

Note that if $0 \in C$ for a co-ideal C, then $C = A$. Now consider the smallest co-ideal $C(\{1\})$ that contains 1. An easy exercise (using distributivity) shows that $C(\{1\}) = \{1 + a \mid a \in A\}$. More generally, for every $S \subseteq A$, the smallest co-ideal containing S is

$$C(S) = \{s_1 \cdots s_k + a \mid k \in \mathbb{N}, s_1, \ldots, s_k \in S, a \in A\} \ .$$

If $(A, +, \cdot, 0, 1)$ is a ring, then $0 \in C(\{1\})$, and thus $C(\{1\}) = A$. However, if it is proper, then clearly $0 \notin C(\{1\})$, and thus $C(\{1\}) \neq A$. Now ZORN's Lemma guarantees that in the latter case there exists a maximal co-ideal C such that $C(\{1\}) \subseteq C \subseteq A \setminus \{0\}$. The remaining elements $A \setminus C$ form an ideal that contains 0. This is verified as follows. Let $a \in A$ and $i, i' \in A \setminus C$. Since $i, i' \notin C$ there exist $c, c' \in C$, $n, n' \in \mathbb{N}$, and $b, b' \in A$ such that $i^n c + b = 0$ and $(i')^{n'} c' + b' = 0$ by maximality of C. Indeed, if such elements do not exist, then i or i' can be added to C to induce an even larger, proper co-ideal $C(C \cup \{i\})$ or $C(C \cup \{i'\})$. We show that $ai \notin C$ and $i + i' \notin C$, which proves that $A \setminus C$ is an ideal. Since $(ai)^n c + a^n b = a^n (i^n c + b) = 0$, we have $(ai)^n \notin C$ because $c \in C$. In fact, if $(ai)^n \in C$, then also $(ai)^n c \in C$ and $(ai)^n c + a^n b \in C$, which contradicts $0 \notin C$ because $(ai)^n c + a^n b = 0$. Consequently, $ai \notin C$ because if $ai \in C$, then $(ai)^n \in C$ for every $n \in \mathbb{N}$. Similarly, $(i + (i'))^m cc'$ where $m = n + n'$ can be presented as a sum of elements of the form $a_j = i^j (i')^{m-j} cc'$ where $j \in [m]$. Mind that the same summand can occur multiple times in the sum. Let $j \geq n$. Then $i^j (i')^{m-j} cc' + bi^{j-n} (i')^{m-j} c' = (i^n c + b) i^{j-n} (i')^{m-j} c' = 0$. Let $b_j = bi^{j-n} (i')^{m-j} c'$. An analogous argument applies to the case $j < n$, which yields that $m - j \geq n'$. Thus, for every summand a_j there exists an element $b_j \in A$ such that $a_j + b_j = 0$, which proves that also for $(i + (i'))^m cc'$, which is finite sum of summands a_j, there exists an element $b'' \in A$ such that $(i + (i'))^m cc' + b'' = 0$, which proves that $i + i' \notin C$.

Now we can define the semiring homomorphism h by

$$h(a) = \begin{cases} 1 & \text{if } a \in C \\ 0 & \text{otherwise} \end{cases}$$

for every $a \in A$. Since C is a co-ideal and $A \setminus C$ is an ideal, this mapping h is a semiring homomorphism.

Theorem 1. *For every countably complete semiring $(A, +, \cdot, 0, 1)$ with respect to $\sum$ there exists an essentially complete semiring homomorphism to the* BOOLEAN *semiring, which is countably complete with respect to* max.

Proof. We start with the semiring homomorphism h that we constructed above. Recall that $C \subseteq A \setminus \{0\}$ is maximal co-ideal with $1 \in C$. It remains to prove that h is essentially complete. Let I be an index set, $S \subseteq A$ be finite, and $a_i \in \langle S \rangle$ for every $i \in I$. We have to prove that

$$h\Big(\sum_{i \in I} a_i\Big) = \max_{i \in I} h(a_i) \ , \tag{1}$$

which yields two (mutually exclusive) cases:

- there exists $i \in I$ such that $h(a_i) = 1$ (i.e., $a_i \in C$) or
- $h(a_i) = 0$ (i.e., $a_i \in I$) for all $i \in I$.

In the former case, the right-hand side of (1) evaluates to 1. This yields that we have to prove that $\sum_{i \in I} a_i \in C$. Let $j \in I$ be such that $a_j \in C$. Then $\sum_{i \in I} a_i = a_j + \sum_{i \in I \setminus \{j\}} a_i$ using the axioms (P), (B), and (U). Since $a_j \in C$, also $a_j + \sum_{i \in I \setminus \{j\}} a_i \in C$ because C is a co-ideal, which proves that $\sum_{i \in I} a_i \in C$.

In the second case, the right-hand side of (1) evaluates to 0. Thus, we need to prove that $\sum_{i \in I} a_i \notin C$. Since each a_i is in $\langle S \rangle$, we can represent it as a (finite) sum $\sum_{j=1}^{n_i} b_{ij}$ of products b_{ij} of elements of S as already remarked. Clearly, $b_{ij} \notin C$ for every $1 \leq j \leq n_i$ because otherwise $a_i \in C$. Consequently, we can write

$$\sum_{i \in I} a_i = \sum_{i \in I} \Big(\sum_{j=1}^{n_i} b_{ij} \Big) = \sum_{i \in I'} b_i a'_i$$

for some index set I', and $b_i \in S \setminus C$ and $a'_i \in A$ for every $i \in I'$. The existence of the factors $b_i \notin C$ follows from the fact that $a_i \notin C$. The set $B = \{b_i \mid i \in I'\}$ is finite because $B \subseteq S$. Let $B = \{e_1, \ldots, e_n\}$. Since C is maximal, we know that for each $b \in B$ there exist $\ell_b \in \mathbb{N}$, $c_b \in C$, and $d_b \in A$ such that $b^{\ell_b} c_b + d_b = 0$. Otherwise, the co-ideal $C(C \cup \{b\})$ would still be different from A, which contradicts the maximality of C. By Proposition 1, we know that $b^{\ell_b} c_b = 0$ (and $d_b = 0$). Let $\ell = \sum_{b \in B} \ell_b$ and $c = \prod_{b \in B} c_b$. Then

$$\Big(\sum_{i \in I} a_i \Big)^{\ell} c = \Big(\sum_{i \in I'} b_i a'_i \Big)^{\ell} \cdot \prod_{b \in B} c_b = \sum_{i \in I''} e_1^{\ell_{i1}} \cdots e_n^{\ell_{in}} a''_i c_{e_1} \cdots c_{e_n}$$

for some index set I'', $\ell_{i1}, \ldots, \ell_{in} \in \mathbb{N}$ and $a_i'' \in A$ for every $i \in I''$ such that $\sum_{j=1}^{n} \ell_{ij} = \ell$. From the last expression it is clear that each summand contains a factor $b^m c_b = 0$ for some $b \in B$ and $m \geq \ell_b$. Thus, each summand is 0 and the sum is 0 by Proposition 1, which proves that $(\sum_{i \in I} a_i)^\ell c = 0$. However, since $c \in C$, we proved that $\sum_{i \in I} a_i$ cannot be in C because it would yield $0 \in C$, which is a contradiction. Consequently, $\sum_{i \in I} a_i \notin C$, which proves the statement.

Consequently, we proved that h is $\langle S \rangle$-complete, and since S was chosen arbitrarily, we also proved that h is essentially complete. □

> For the rest of this paper, let $(A, +, \cdot, 0, 1)$ be an arbitrary nontrivial (i.e., $0 \neq 1$) commutative semiring.

2.2 Sets, relations, and trees

We denote the set of all nonnegative integers (including 0) by $\mathbb{N}$. For every $n \in \mathbb{N}$, the subset $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ is denoted by $[n]$. We fix the set $\mathrm{X} = \{\mathrm{x}_1, \mathrm{x}_2, \ldots\}$ of (formal) variables and let $\mathrm{X}_n = \{\mathrm{x}_i \mid i \in [n]\}$ for every $n \in \mathbb{N}$.

Now, let S, T, and U be countable sets. A *relation from S to T* is a subset of $S \times T$. Let $R \subseteq S \times T$ and $R' \subseteq T \times U$. The *inverse relation of R*, denoted by R^{-1}, is $\{(t, s) \mid (s, t) \in R\}$ and the *composition of R and R'*, denoted by $R;R'$, is $\{(s, u) \mid \exists t \in T\colon (s, t) \in R, (t, u) \in R'\}$. These notions extend to classes of relations in the standard manner. A *relation on S* is a subset of $S \times S$. For every set $L \subseteq S$ we denote by id_L the relation $\{(s, s) \mid s \in L\}$. The reflexive and transitive closure of a relation $R \subseteq S \times S$ is denoted by R^*.

Next, we extend these notions to the weighted setting. A *weighted relation from S to T* is a mapping of $\rho\colon S \times T \to A$. Let $\rho\colon S \times T \to A$ and $\rho'\colon T \times U \to A$. The *inverse relation of ρ*, denoted by ρ^{-1}, is such that $\rho^{-1}(t, s) = \rho(s, t)$ for every $s \in S$ and $t \in T$, and the *composition of ρ and ρ'*, denoted by $\rho\,;\rho'$, is such that $(\rho\,;\rho')(s, u) = \sum_{t \in T} \rho(s, t)\rho'(t, u)$ for every $s \in S$ and $u \in U$. Depending on the set T and the weighted relations ρ and ρ', the sum in the definition of $\rho;\rho'$ might be infinite. If it is, then we typically assume that $(A, +, \cdot, 0, 1)$ is countably complete with respect to $\sum$. We will discuss this issue in more detail later on. Again, these notions extend to classes of weighted relations in the standard manner. A *weighted relation on S* is a mapping $\rho\colon S \times S \to A$. For every weighted set $\varphi\colon S \to A$, we denote by id_φ the weighted relation such that $\mathrm{id}_\varphi(s, s) = \varphi(s)$ and $\mathrm{id}_\varphi(s, s') = 0$ for every $s, s' \in S$ such that $s \neq s'$. If $\varphi(s) = 1$ for every $s \in S$, then we also just write id instead of id_φ. Given that $(A, +, \cdot, 0, 1)$ is countably complete with respect to $\sum$, the reflexive and transitive closure of a weighted relation $\rho\colon S \times S \to A$ is denoted by ρ^* and is defined by $\rho^*(s, s') = \sum_{n \in \mathbb{N}} \rho^n(s, s')$ where $\rho^0 = \mathrm{id}$ and $\rho^{k+1} = \rho\,;\rho^k$ for every $k \in \mathbb{N}$. Note that all weighted notions over the BOOLEAN semiring correspond to the unweighted notions via the mapping 'supp'. For example, the weighted relation $\rho\colon S \times T \to \{0, 1\}$ corresponds to the (unweighted) relation $\mathrm{supp}(\rho)$.

The set of all finite sequences (words) over S is denoted by S^*, of which ε denotes the empty sequence (the empty word). The concatenation of the words $v, w \in S^*$

is denoted by $v.w$ or simply by vw. The length of a word $w \in S^*$ (i.e., the number of occurrences of elements of S in w) is denoted by $|w|$.

An *alphabet* Σ is a nonempty and finite set, of which the elements are called *symbols*. Next, we define trees using only alphabets. In contrast to many definitions in the literature [12, 24, 25, 23], we do not assume a ranked alphabet, which means that a symbol can have different numbers of children in a tree. This is only a simplification because our automata model will still have only a finite number of rules, which thus determine a maximal rank for each used symbol. Let Q be an alphabet and L a countable set of leaf labels. For every set T, we let

$$Q(T) = \{q(t) \mid q \in Q, t \in T\} \ .$$

The set $T_\Sigma(L)$ of Σ-*trees with leaf labels* L is the smallest set T such that $L \subseteq T$ and $\sigma(t_1, \ldots, t_k) \in T$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T$. We generally assume that $\Sigma \cap L = \emptyset$, and thus we write $\alpha()$ simply as α for every $\alpha \in \Sigma$. Given another alphabet Δ and $T \subseteq T_\Delta(L)$, we treat elements of $T_\Sigma(T)$ and $Q(T)$ as particular trees of $T_{Q \cup \Sigma \cup \Delta}(L)$. For every $\gamma \in \Sigma$, we abbreviate the tree $\gamma(\gamma(\cdots\gamma(t)\cdots))$ with n symbols γ on top of $t \in T_\Sigma(L)$ simply by $\gamma^n(t)$. Finally, we write T_Σ for $T_\Sigma(\emptyset)$. Note that $T_\Sigma(L)$ is countable and that the elements of L can only appear as leaves in trees of $T_\Sigma(L)$.

The set $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ of *positions* of a tree $t \in T_\Sigma(L)$ is inductively defined by $\mathrm{pos}(\ell) = \{\varepsilon\}$ for every $\ell \in L$ and

$$\mathrm{pos}(\sigma(t_1, \ldots, t_k)) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in \mathrm{pos}(t_i)\}$$

for every $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(L)$. Let $t, t' \in T_\Sigma(L)$ and $w \in \mathrm{pos}(t)$. The label of t at position w is $t(w)$, and the subtree of t that is rooted at w is $t|_w$. We can define these notions inductively as follows: $\ell(\varepsilon) = \ell|_\varepsilon = \ell$ for every $\ell \in L$ and

$$t(w) = \begin{cases} \sigma & \text{if } w = \varepsilon \\ t_i(v) & \text{if } w = iv \text{ with } i \in [k] \end{cases} \quad \text{and} \quad t|_w = \begin{cases} t & \text{if } w = \varepsilon \\ t_i|_v & \text{if } w = iv \text{ with } i \in [k] \end{cases}$$

where $t = \sigma(t_1, \ldots, t_k)$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(L)$. For every set of labels $S \subseteq \Sigma \cup L$, we let $\mathrm{pos}_S(t) = \{w \in \mathrm{pos}(t) \mid t(w) \in S\}$. For $S = \{s\}$ we abbreviate $\mathrm{pos}_S(t)$ simply by $\mathrm{pos}_s(t)$. We say that $s \in S$ *occurs* $|\mathrm{pos}_s(t)|$ times in t. Finally, $t[u]_w$ denotes the tree that is obtained from $t \in T_\Sigma(L)$ by replacing the subtree $t|_w$ at w by $u \in T_\Delta(L)$.

The *height* $\mathrm{ht}(t)$ of t is $\max\{|w| + 1 \mid w \in \mathrm{pos}(t)\}$, and the *size* $|t|$ of the tree t is $|t| = |\mathrm{pos}(t)|$. Recall the special set X of formal variables. We let $\mathrm{var}(t) = \{x \in \mathrm{X} \mid \mathrm{pos}_x(t) \neq \emptyset\}$ for every $t \in T_\Sigma(L \cup \mathrm{X})$. The tree t is *linear* (respectively, *nondeleting*) in $V \subseteq \mathrm{X}$, if every $x \in V$ occurs at most (respectively, at least) once in t. Every $t \in T_\Sigma(V)$ that is linear and nondeleting in V is a V-*context* of $T_\Sigma(V)$. The set of all V-contexts of $T_\Sigma(V)$ is denoted by $C_\Sigma(V)$. For every such context and $x \in \mathrm{var}(t)$, we identify the unique element of $\mathrm{pos}_x(t)$ with $\mathrm{pos}_x(t)$. If t is linear and nondeleting in X_k for some $k \in \mathbb{N}$ and the variables occur in order (i.e., $\mathrm{pos}_{x_i}(t) < \mathrm{pos}_{x_j}(t)$ in the usual lexicographic ordering

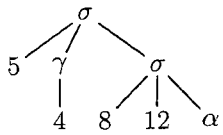


Figure 1: Graphical representation of the tree $\sigma(5, \gamma(4), \sigma(8, 12, \alpha))$.

for all $1 \leq i < j \leq k$), then t is called *normalized*. For every $V \subseteq \mathrm{X}$, a mapping $\theta\colon V \to T_\Sigma(L)$ is a *substitution*. The substitution θ can be applied to a tree $t \in T_\Sigma(L \cup \mathrm{X})$, written $t\theta$, which yields the tree that is obtained by replacing (in parallel) all occurrences of a variable $x \in V$ by $\theta(x)$. Formally, $x\theta = \theta(x)$ for every $x \in V$, and $\sigma(t_1, \ldots, t_k)\theta = \sigma(t_1\theta, \ldots, t_k\theta)$ for every $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(L \cup \mathrm{X})$. To avoid explicitly defining a substitution $\theta\colon \mathrm{X}_k \to T_\Sigma(L)$, we sometimes write $t[\theta(\mathrm{x}_1), \ldots, \theta(\mathrm{x}_k)]$ for $t\theta$.

Example 1. Let $\Sigma = \{\sigma, \gamma, \alpha\}$. Then $t = \sigma(5, \gamma(4), \sigma(8, 12, \alpha))$ is a tree of $T_\Sigma(\mathbb{N})$. Its graphical representation is displayed in Figure 1. Its set $\mathrm{pos}(t)$ of positions is $\{\varepsilon, 1, 2, 2.1, 3, 3.1, 3.2, 3.3\}$ and $\mathrm{pos}_\sigma(t) = \{\varepsilon, 3\}$. The tree $\sigma(\mathrm{x}_1, \mathrm{x}_3, \mathrm{x}_2, \mathrm{x}_2)$ is linear and nondeleting in $\{\mathrm{x}_1, \mathrm{x}_3\}$, but not linear in X_3.

3 The Model

In this section, we will recall the main model of this survey: the weighted extended top-down tree transducer [2, 3, 38, 40]. However, we first recall the corresponding automaton model: the weighted tree automaton [6, 34, 17, 23]. A *weighted tree language* (or tree series) is simply a weighted set of trees; i.e., a mapping $\varphi\colon T_\Sigma \to A$ for some alphabet Σ.

Definition 1. *A* weighted (bottom-up) tree automaton *(wta) is a tuple* (Q, Σ, δ, F) *such that*

- Q *is an alphabet of* states,
- Σ *is an alphabet of* input symbols *such that* $Q \cap \Sigma = \emptyset$,
- $\delta\colon Q^* \times \Sigma \times Q \to A$ *is a* transition weight mapping *with finite* $\mathrm{supp}(\delta)$, *and*
- $F \subseteq Q$ *is a set of* final states.

The transition weight mapping δ of the wta $M = (Q, \Sigma, \delta, F)$ is extended to a mapping $\underline{\delta}\colon T_\Sigma \times Q \to A$ as follows:

$$\underline{\delta}(\sigma(t_1, \ldots, t_k), q) = \sum_{q_1, \ldots, q_k \in Q} \delta(q_1 \cdots q_k, \sigma, q) \cdot \prod_{i=1}^{k} \underline{\delta}(t_i, q_i)$$

for every $q \in Q$, $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma$. In the sequel, we will simply write δ instead of $\underline{\delta}$. The weighted tree language $\varphi_M\colon T_\Sigma \to A$ *recognized by* M is such that $\varphi_M(t) = \sum_{q \in F} \delta(t, q)$ for every $t \in T_\Sigma$. A weighted tree language $\varphi\colon T_\Sigma \to A$ is *recognizable* if there exists a wta M such that $\varphi_M = \varphi$.

The recognizable weighted tree languages are the natural generalization of the recognizable tree languages [24, 25]. For the BOOLEAN semiring the two notions coincide. An excellent introduction into the subject is presented in [23]. Here we just present a quick example.

Example 2. Let us consider the artic semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ and the wta (Q, Σ, δ, F) with

- $Q = \{z, h\}$ and $F = \{h\}$,
- $\Sigma = \{\sigma, \alpha\}$, and
- δ returns $-\infty$ except in the following cases:

$$\begin{array}{lll} \delta(\varepsilon, \alpha, z) = 0 & \delta(z, \alpha, z) = 0 & \delta(zz, \sigma, z) = 0 \\ \delta(\varepsilon, \alpha, h) = 1 & \delta(h, \alpha, h) = 1 & \delta(zh, \sigma, h) = 1 \\ & & \delta(hz, \sigma, h) = 1 \ . \end{array}$$

This wta rejects (i.e., assigns weight $-\infty$ to) all trees t that contain a symbol α with at least two children [i.e., trees t that have a position $w \in \mathrm{pos}_\alpha(t)$ such that $a.2$ is also in $\mathrm{pos}(t)$] or a symbol σ with anything but 2 children. Moreover, it assigns the height $\mathrm{ht}(t)$ to the remaining trees t.

Next, we recall the weighted extended top-down tree transducer [8, 2, 27, 33]. For simplicity, we will henceforth just call them "extended tree transducer" (xtt). We essentially follow the definitions of [38, 41], in which the corresponding unweighted device is discussed in detail. We will lift the results obtained in [38, 41] to the weighted setting either directly or using the semiring homomorphism introduced in Section 2. We start with the definition of the general device.

Definition 2. *A* (weighted) extended (top-down) tree transducer *(xtt) is a tuple* $M = (Q, \Sigma, \Delta, I, R)$ *where*

- Q *is an alphabet of* states,
- Σ *and* Δ *are alphabets of* input *and* output symbols *such that* $Q \cap (\Sigma \cup \Delta) = \emptyset$,
- $I \subseteq Q$ *is a set of* initial states, *and*
- $R\colon Q(T_\Sigma(\mathrm{X})) \times T_\Delta(Q(\mathrm{X})) \to A$ *assigns* rule weights *such that* $\mathrm{supp}(R)$ *is finite,* l *is linear in* X *and* $\mathrm{var}(r) \subseteq \mathrm{var}(l)$ *for every* $(l, r) \in \mathrm{supp}(R)$.

If for every $(l, r) \in \mathrm{supp}(R)$ *there exist* $k \in \mathbb{N}$, $q \in Q$, *and* $\sigma \in \Sigma$ *such that* $l = q(\sigma(\mathrm{x}_1, \ldots, \mathrm{x}_k))$, *then* M *is a* top-down tree transducer *[35, 14].*

In the sequel, we often write $l \to r \in \mathrm{supp}(R)$ instead of $(l, r) \in \mathrm{supp}(R)$, and we write $l \xrightarrow{a} r \in R$ instead of $R(l, r) = a$. The xtt M is *linear* (respectively, *nondeleting* if r is linear (respectively, nondeleting) in $\mathrm{var}(l)$ for every $l \to r \in \mathrm{supp}(R)$. A rule of the form $q(x) \to r$ with $q \in Q$, $x \in \mathrm{X}$, and $r \in T_\Delta(Q(\mathrm{X}))$ is called *input* ε*-rule*, and any rule of the form $l \to q(x)$ with $q \in Q$, $x \in \mathrm{X}$, and $l \in Q(T_\Sigma(\mathrm{X}))$ is called *output* ε*-rule*. A rule that is both an input and an output ε-rule is called a *pure* ε*-rule*. The set of all pure ε-rules of $\mathrm{supp}(R)$ is denoted by R^ε. Any remaining rule contains at least one input or output symbol.

Example 3 (see [41, Section 3]). Let us consider the field $(\mathbb{R}, +, \cdot, 0, 1)$ of real numbers. The xtt $(Q, \Sigma, \Delta, \{q\}, R)$ with

$$\begin{aligned} Q &= \{q, q_{\mathrm{S}}, q_{\mathrm{V}}, q_{\mathrm{NP}}\} \\ \Sigma &= \Gamma \cup \{saw, the, boy, door\} \\ \Delta &= \Gamma \cup \{ra'aa, atefl, albab\} \\ \Gamma &= \{\mathrm{CONJ}, \mathrm{S}, \mathrm{VP}, \mathrm{V}, \mathrm{DT}, \mathrm{NP}, \mathrm{N}\} \end{aligned}$$

and the following weighted rules of R

$$q(\mathrm{x}_1) \xrightarrow{.2} q_{\mathrm{S}}(\mathrm{x}_1) \qquad (\rho_1)$$

$$q(\mathrm{x}_1) \xrightarrow{.8} \mathrm{S}(\mathrm{CONJ}(wa\text{-}), q_{\mathrm{S}}(\mathrm{x}_1)) \qquad (\rho_2)$$

$$q_{\mathrm{S}}(\mathrm{S}(\mathrm{x}_1, \mathrm{VP}(\mathrm{x}_2, \mathrm{x}_3))) \xrightarrow{1} \mathrm{S}'(q_{\mathrm{V}}(\mathrm{x}_2), q_{\mathrm{NP}}(\mathrm{x}_1), q_{\mathrm{NP}}(\mathrm{x}_3)) \qquad (\rho_3)$$

$$q_{\mathrm{V}}(\mathrm{V}(saw)) \xrightarrow{.7} \mathrm{V}(ra'aa) \qquad (\rho_4)$$

$$q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(the), \mathrm{N}(boy))) \xrightarrow{.6} \mathrm{NP}(\mathrm{N}(atefl)) \qquad (\rho_5)$$

$$q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(the), \mathrm{N}(door))) \xrightarrow{.5} \mathrm{NP}(\mathrm{N}(albab) \qquad (\rho_6)$$

is linear and nondeleting. In addition, it has 2 input ε-rules, 1 output ε-rule, and 1 pure ε-rule. It is not a top-down tree transducer.

Our reference semantics of the xtt $M = (Q, \Sigma, \Delta, I, R)$ is given by rewriting [13, 22, 41]. Let $\zeta, \xi \in T_\Delta(Q(T_\Sigma))$ and $\rho = (l \to r) \in \mathrm{supp}(R)$. The *leftmost redex* in ζ is the least position $w \in \mathrm{pos}_Q(\zeta)$ with respect to the lexicographic (total) ordering on $\mathbb{N}^*$. In other words, the leftmost redex is the leftmost position in the tree that is labeled with a state. We say that ζ *rewrites to* ξ *using* ρ, denoted by $\zeta \Rightarrow_M^\rho \xi$, if there exist a minimal position $w \in \mathrm{pos}_Q(\zeta)$ and a substitution $\theta\colon \mathrm{var}(l) \to T_\Sigma$ such that $\zeta|_w = l\theta$ and $\xi = \zeta[r\theta]_w$. Intuitively, we identify a rule, of which the left-hand side matches the subtree at the leftmost redex, and then we replace this subtree by the corresponding instantiated right-hand side. The *weighted relation* τ_M (or weighted tree transformation) *computed by* M is given by

$$\tau_M(t, u) = \sum_{\substack{q \in I, k \in \mathbb{N}, \rho_1, \ldots, \rho_k \in \mathrm{supp}(R) \\ q(t) \Rightarrow_M^{\rho_1} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_1) \cdots R(\rho_k) \qquad (2)$$

for every $t \in T_\Sigma$ and $u \in T_\Delta$. Two xtt M and M' are *equivalent* if $\tau_M = \tau_{M'}$. The sum in (2) can be infinite. Let us present a short example before we discuss the finiteness of the sum of (2) in some detail. The semantics shows that variables can be consistently renamed without effect. Consequently, for every xtt $M = (Q, \Sigma, \Delta, I, R)$ there exists an equivalent xtt $M' = (Q, \Sigma, \Delta, I, R')$ such that for every $(l, r) \in \mathrm{supp}(R')$ there exists $k \in \mathbb{N}$ with $\mathrm{var}(l) = \{\mathrm{x}_1, \ldots, \mathrm{x}_k\}$. In the following, we will silently assume this normal form.

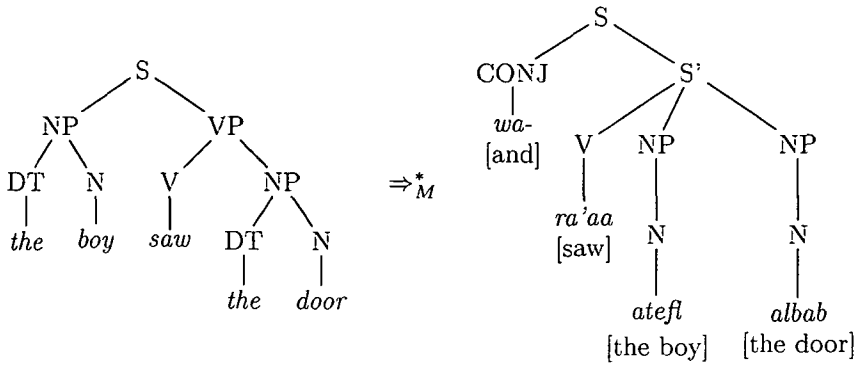


Figure 2: English-to-Arabic translation on syntax trees.

Example 4. Let us reconsider the xtt M of Example 3, and let

$$t = \mathrm{S}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy})), \mathrm{VP}(\mathrm{V}(\mathit{saw}), \mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))) \ ,$$

which is depicted in Figure 2. Then

$$\begin{aligned}
q(t) &\Rightarrow^{\rho_2}_M \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), q_{\mathrm{S}}(t)) \\
&\Rightarrow^{\rho_3}_M \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(q_{\mathrm{V}}(\mathrm{V}(\mathit{saw})), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))), \\
&\qquad\qquad q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))) \\
&\Rightarrow^{\rho_4}_M \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))), \\
&\qquad\qquad q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))) \\
&\Rightarrow^{\rho_5}_M \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{atefl})), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door}))))) \\
&\Rightarrow^{\rho_6}_M \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{atefl})), \mathrm{NP}(\mathrm{N}(\mathit{albab})))) = u \ .
\end{aligned}$$

Since this is the only derivation from $q(t)$ to u, we conclude that

$$\tau_M(t, u) = R(\rho_2)R(\rho_3)R(\rho_4)R(\rho_5)R(\rho_6) = 0.8 \cdot 1 \cdot 0.7 \cdot 0.6 \cdot 0.5 = 0.168 \ .$$

For illustration, t and u are displayed in Figure 2.

Let us return to the sum in (2). Clearly, the length k of the derivation is at most $|t|+|u|$ if no pure ε-rule is used (i.e., if $\rho_i \notin R^\epsilon$ for all $i \in [k]$). In this case, the sum in (2) is finite, which yields that the semantics of any top-down tree transducer is well-defined. In the presence of pure ε-rules, we assume that $(A, +, \cdot, 0, 1)$ is countably complete with respect to $\sum$. Consequently, the sum is well-defined. We will get rid of this case distinction after an important characterization result that relates linear nondeleting xtt to another device that computes weighted tree transformations: the weighted linear complete bimorphism.

A *linear and complete tree homomorphism* $f \colon T_\Gamma \to T_\Sigma$ is such that for every $k \in \mathbb{N}$ and $\gamma \in \Gamma$ there exists $f_k(\gamma) \in C_\Sigma(\mathrm{X}_k)$ such that

$$f(\gamma(s_1, \ldots, s_k)) = f_k(\gamma)[f(s_1), \ldots, f(s_k)] \ .$$

If additionally $f_k(\gamma) \neq x_1$ for every $k \in \mathbb{N}$ and $\gamma \in \Gamma$, then f is *ε-free.*

Since wta are rather unsuitable for the next result, we use another model: the *weighted regular tree grammar* [1]. Such a grammar is tuple $G = (N, \Sigma, S, P)$ where N is a finite set of nonterminals, Σ is an alphabet of input symbols, $S \subseteq N$ is a set of start symbols, and $P\colon N \times T_\Sigma(N) \to A$ assigns weights to productions such that $\mathrm{supp}(P)$ is finite. It computes in a step-wise fashion. Given $\xi \in T_\Sigma(N)$, let $w \in \mathrm{pos}_N(\xi)$ be the leftmost position (i.e., the smallest of $\mathrm{pos}_N(\xi)$ with respect to the lexicographic ordering on $\mathbb{N}^*$). If there exists a production $\rho = (n, s) \in \mathrm{supp}(P)$ such that $n = \xi(w)$, then we write $\xi \Rightarrow_G^\rho \zeta$ where $\zeta = \xi[s]_w$. In other words, the nonterminal is replaced by the corresponding right-hand side of the production. The semantics of G is then given by

$$\varphi_G(t) = \sum_{\substack{s \in S, k \in \mathbb{N}, \rho_1, \ldots, \rho_k \in \mathrm{supp}(P) \\ s \Rightarrow_G^{\rho_1} \cdots \Rightarrow_G^{\rho_k} t}} P(\rho_1) \cdots P(\rho_k)$$

for every $t \in T_\Sigma$. To avoid a discussion of infinite summations here, we assume that $s \notin N$ for every $(n, s) \in \mathrm{supp}(P)$, which guarantees that the above sum is finite. It is known that such weighted regular tree grammars also compute the recognizable weighted tree languages [1, Proposition 3.1] (also see [23] for a detailed account). The following important result is well-known from the literature [35, 17].

Theorem 2 (see [35] and [17, Corollary 6.10]). *For every linear, complete, and ε-free tree homomorphism $f\colon T_\Gamma \to T_\Sigma$ and every recognizable weighted tree language $\psi\colon T_\Gamma \to A$, the weighted tree language $\varphi\colon T_\Sigma \to A$, which is given by $\varphi(t) = \sum_{s \in T_\Gamma, f(s) = t} \psi(s)$, is again recognizable.*

Proof. Intuitively, we translate a symbol $\gamma \in \Gamma$ with the help of f into a context, which we then process in a single step charging the original weight. Now let us construct a weighted regular tree grammar for φ. Formally, let $M = (Q, \Gamma, \delta, F)$ be a wta recognizing ψ. The weighted regular tree grammar $G = (Q, \Sigma, F, P)$ is such that

$$P(q, s) = \sum_{\substack{k \in \mathbb{N}, q_1, \ldots, q_k \in Q \\ s = f_k(\gamma)[q_1, \ldots, q_k]}} \delta(q_1 \cdots q_k, \gamma, q)$$

for every $q \in Q$ and $s \in T_\Sigma(Q)$. Clearly, $s \notin Q$ for every $(q, s) \in \mathrm{supp}(P)$. Moreover, it should be clear that $\varphi_G = \varphi$, which proves that φ is recognizable. □

A *weighted linear and complete bimorphism* [3, 21] is a tuple $B = (f, \varphi, g)$ such that $f\colon T_\Gamma \to T_\Sigma$ and $g\colon T_\Gamma \to T_\Delta$ are linear and complete tree homomorphisms and $\varphi\colon T_\Gamma \to A$ is recognizable. The weighted tree transformation *computed by* B is

$$\tau_B(t, u) = \sum_{\substack{s \in T_\Gamma \\ f(s) = t, g(s) = u}} \varphi(s)$$

for every $t \in T_\Sigma$ and $u \in T_\Delta$.

Theorem 3 (see [38, Theorem 4] and [21]). *For every linear and nondeleting xtt there exists an equivalent weighted linear and complete bimorphism and vice versa.*

Proof. We have to prove both directions. Let $B = (f, \varphi, g)$ be a weighted linear and complete bimorphism with $f\colon T_\Gamma \to T_\Sigma$ and $g\colon T_\Gamma \to T_\Delta$. Moreover, let $N = (Q, \Gamma, \delta, F)$ be a wta such that $\varphi_N = \varphi$. Roughly speaking, we use the control structure of N as control structure of the xtt M that we construct and use f and g to determine the left- and right-hand sides of the rules, respectively. Formally, we construct the linear nondeleting xtt $M = (Q, \Sigma, \Delta, F, R)$ as follows. For every $l \in Q(T_\Sigma)$ and $r \in T_\Delta(Q(\mathrm{X}))$, let

$$R(l,r) = \sum_{\substack{(q_1 \cdots q_k, \gamma, q) \in \operatorname{supp}(\delta) \\ l = q(f_k(\gamma)), r = g_k(\gamma)[q_1(\mathrm{x}_1), \ldots, q_k(\mathrm{x}_k)]}} \delta(q_1 \cdots q_k, \gamma, q) \ .$$

Note that M is linear and nondeleting.

For the converse, let a linear and nondeleting xtt $M = (Q, \Sigma, \Delta, I, R)$ be given. Without loss of generality, we suppose that for every $l \to r \in \operatorname{supp}(R)$ there exists $k \in \mathbb{N}$ with $\operatorname{var}(l) = \{\mathrm{x}_1, \ldots, \mathrm{x}_k\}$. We construct f, g, and a wta $N = (Q, \Gamma, \delta, I)$ with $\Gamma = \operatorname{supp}(R)$ as follows. For every $\rho \in \operatorname{supp}(R)$, we have $\rho = q(l) \to r\theta$ with $l \in C_\Sigma(\mathrm{X}_k)$, $r \in C_\Delta(\mathrm{X}_k)$, and $q, q_1, \ldots, q_k \in Q$ where θ is the substitution such that $\mathrm{x}_i\theta = q_i(\mathrm{x}_i)$ for every $i \in [k]$. For this rule ρ, let $f_k(\rho) = l$, $g_k(\rho) = r$, and $\delta(q_1 \cdots q_k, \rho, q) = R(\rho)$. The remaining values of $f_k(\gamma)$ and $g_k(\gamma)$ are irrelevant and all unmentioned values of δ are 0.

For both directions it remains to prove that $\tau_M = \tau_B$. To this end, it can be shown for every $q \in Q$, $t \in T_\Sigma$, and $u \in T_\Delta$ that

$$\sum_{\substack{n \in \mathbb{N}, \rho_1, \ldots, \rho_n \in \operatorname{supp}(R) \\ q(t) \Rightarrow_M^{\rho_1} \cdots \Rightarrow_M^{\rho_n} u}} R(\rho_1) \cdots R(\rho_n) = \sum_{\substack{s \in T_\Gamma \\ f(s) = t, g(s) = u}} \delta(s, q) \ .$$

The proof of that statement is omitted here. The unweighted case is proved in [38, Theorem 4] and the weighted case is discussed in [20, 21]. □

It follows immediately from Theorem 3 that linear and nondeleting xtt are symmetric; i.e., for every linear and nondeleting xtt $M = (Q, \Sigma, \Delta, I, R)$ there exists a linear and nondeleting xtt M' such that $\tau_{M'}(u,t) = \tau_M(t,u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. This property is not quite obvious from the definition of such xtt, but can trivially be observed on the bimorphism representation.

Now, we will eliminate pure ε-rules in the standard manner in order to avoid infinite sums, which only occur in the semantics of xtt with pure ε-rules. To this end, let $(A, +, \cdot, 0, 1)$ be countably complete with respect to $\sum$, and let

$$E_{p,q} = \sum_{\substack{l \to r \in R^\varepsilon \\ l(\varepsilon) = p, r(\varepsilon) = q}} R(l,r)$$

for every $p, q \in Q$. Using the matrix E^*, which is well-defined due to the countable completeness, we can construct the equivalent xtt $M' = (Q, \Sigma, \Delta, I, R')$ such that

$$R'(l, r) = 0 \qquad \text{for all } l, r \in Q(\mathrm{X})$$
$$R'(p(\ell), r) = \sum_{q \in Q} E^*_{p,q} R(q(\ell), r) \qquad \text{for all } p, q \in Q,\ \ell \in T_\Sigma(\mathrm{X}), \text{ and } r \in T_\Delta(Q(\mathrm{X}))\ .$$

In fact, the countable completeness is only required if there are cyclic pure ε-rules. We omit the proof that M and M' are equivalent. Clearly, the xtt M' has no pure ε-rules.

Theorem 4. *If $(A, +, \cdot, 0, 1)$ is countably complete with respect to $\sum$, then for every xtt we can construct an equivalent xtt without pure ε-rules.*

Example 5. Recall the xtt $(Q, \Sigma, \Delta, \{q\}, R)$ of Example 3, which has the pure ε-rule ρ_1. Pure ε-rule elimination as outlined above yields the xtt with the rules (ρ_2)–(ρ_6) with their original weight and the new rule

$$q(\mathrm{S}(\mathrm{x}_1, \mathrm{VP}(\mathrm{x}_2, \mathrm{x}_3))) \xrightarrow{\cdot 2} \mathrm{S}'(q_{\mathrm{V}}(\mathrm{x}_2), q_{\mathrm{NP}}(\mathrm{x}_1), q_{\mathrm{NP}}(\mathrm{x}_3))\ . \qquad (\rho_3')$$

For the rest of the section, we will assume that all used xtt do not have pure ε-rules. This assumption is often made immediately in the literature [40, 20] to ensure that the sum in (2) is always well-defined. The countable completeness of the semiring is thus only needed in the elimination of the pure ε-rules. The class of weighted tree transformations computed by xtt is denoted by XTOP. The subclasses computed by linear and linear nondeleting xtt are denoted by l-XTOP and ln-XTOP, respectively. The corresponding classes of weighted tree transformations computed by top-down tree transducers are TOP, l-TOP, and ln-TOP.

The rewrite semantics is very illustrative, but difficult to handle in proofs due to its essentially non-recursive specification. Next, we are going to present an alternative way to recursively define the semantics and then show that for every xtt both semantics indeed define the same weighted tree transformation. We need one additional notion. Let Σ be an alphabet and $t \in T_\Sigma$. Then

$$\mathrm{match}(t) = \{(c, \theta) \mid k \in \mathbb{N}, c \in C_\Sigma(\mathrm{X}_k), \theta \colon \mathrm{X}_k \to T_\Sigma \text{ with } t = c\theta\}\ .$$

Note that $\mathrm{match}(t)$ is finite. Recall that a normalized tree is linear and nondeleting in X_k for some $k \in \mathbb{N}$ and its variables $\{\mathrm{x}_1, \ldots, \mathrm{x}_k\}$ occur in order.

Definition 3. *Let $M = (Q, \Sigma, \Delta, I, R)$ be an xtt (without pure ε-rules). We define the mapping $h_R \colon Q(T_\Sigma) \times T_\Delta \to A$ for every $\xi \in Q(T_\Sigma)$ and $u \in T_\Delta$ by*

$$h_R(\xi, u) = \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l, \theta') \in \mathrm{match}(\xi) \\ (s, \theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta \colon \mathrm{var}(s) \to Q(\mathrm{var}(l)) \\ r = s\theta}} R(l, r) \cdot \prod_{x \in \mathrm{var}(s)} h_R(x\theta\theta', x\theta'')\ .$$

Note that this recursion is well-defined because $|x\theta\theta'| \leq |t|$ and $|x\theta''| \leq |u|$ and one of the inequalities is strict due to the fact that $\{l, r\} \not\subseteq Q(\mathrm{X})$ for every $l \to r \in \mathrm{supp}(R)$. Consequently, we can define the weighted tree transformation $\tau'_M : T_\Sigma \times T_\Delta \to A$ by

$$\tau'_M(t, u) = \sum_{q \in I} h_R(q(t), u)$$

for every $t \in T_\Sigma$ and $u \in T_\Delta$.

Theorem 5. *For every xtt M (without pure ε-rules) we have $\tau_M = \tau'_M$.*

Proof. The statement follows immediately from the following statement, which we prove by induction on $|\xi| + |u|$.

$$h_R(\xi, u) = \sum_{\substack{k \in \mathbb{N}, \rho_1, \ldots, \rho_k \in \mathrm{supp}(R) \\ \xi \Rightarrow_M^{\rho_1} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_1) \cdots R(\rho_k)$$

for every $\xi \in Q(T_\Sigma)$, and $u \in T_\Delta$.

$$\begin{aligned}
&\sum_{\substack{k \in \mathbb{N}, \rho_1, \ldots, \rho_k \in \mathrm{supp}(R) \\ \xi \Rightarrow_M^{\rho_1} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_1) \cdots R(\rho_k) \\
&= \sum_{\substack{k \in \mathbb{N}, \rho_1, \ldots, \rho_k \in \mathrm{supp}(R) \\ \rho_1 = l \to r, \theta \colon \mathrm{var}(l) \to T_\Sigma, \xi = l\theta \\ \xi \Rightarrow_M^{\rho_1} r\theta \Rightarrow_M^{\rho_2} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_1) \cdots R(\rho_k) \\
&= \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l, \theta) \in \mathrm{match}(\xi)}} R(l, r) \cdot \Big(\sum_{\substack{k \in \mathbb{N}, \rho_2, \ldots, \rho_k \in \mathrm{supp}(R) \\ r\theta \Rightarrow_M^{\rho_2} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_2) \cdots R(\rho_k) \Big) \\
&= \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l, \theta') \in \mathrm{match}(\xi) \\ (s, \theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta \colon \mathrm{var}(s) \to Q(\mathrm{X}) \\ r = s\theta}} R(l, r) \cdot \prod_{w \in \mathrm{pos}_Q(r)} \Big(\sum_{\substack{n \in \mathbb{N}, \rho'_1, \ldots, \rho'_n \in \mathrm{supp}(R) \\ r\theta'|_w \Rightarrow_M^{\rho'_1} \cdots \Rightarrow_M^{\rho'_n} u|_w}} R(\rho'_1) \cdots R(\rho'_n) \Big) \\
&= \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l, \theta') \in \mathrm{match}(\xi) \\ (s, \theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta \colon \mathrm{var}(s) \to Q(\mathrm{X}) \\ r = s\theta}} R(l, r) \cdot \prod_{x \in \mathrm{var}(s)} \Big(\sum_{\substack{n \in \mathbb{N}, \rho'_1, \ldots, \rho'_n \in \mathrm{supp}(R) \\ x\theta\theta' \Rightarrow_M^{\rho'_1} \cdots \Rightarrow_M^{\rho'_n} x\theta''}} R(\rho'_1) \cdots R(\rho'_n) \Big) \\
&\overset{\text{I.H.}}{=} \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l, \theta') \in \mathrm{match}(\xi) \\ (s, \theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta \colon \mathrm{var}(s) \to Q(\mathrm{X}) \\ r = s\theta}} R(l, r) \cdot \prod_{x \in \mathrm{var}(s)} h_R(x\theta\theta', x\theta'')
\end{aligned}$$

$$= h_R(\xi, u) \ ,$$

where we isolated the first derivation step in the first 2 steps, split the subderivations in the third step, and simplified the obtained expression in the fourth step before we used the induction hypothesis in the fifth step. □

In the following, we will often use this alternative semantics of xtt to prove the correctness of constructions. We will not explicitly recall that it yields the same results as our reference semantics based on rewriting.

Example 6. Let us reconsider the xtt M of Example 5, and let

$$\begin{aligned} t &= \mathrm{S}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy})), \mathrm{VP}(\mathrm{V}(\mathit{saw}), \mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))) \\ u &= \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{atefl})), \mathrm{NP}(\mathrm{N}(\mathit{albab})))) \\ u' &= \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{atefl})), \mathrm{NP}(\mathrm{N}(\mathit{albab}))) \end{aligned}$$

as in Example 4. Then

$$\begin{aligned} h_R(q(t), u) &= \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l,\theta') \in \mathrm{match}(\xi) \\ (s,\theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta\colon \mathrm{var}(s) \to Q(\mathrm{var}(l)) \\ r = s\theta}} R(l,r) \cdot \prod_{x \in \mathrm{var}(s)} h_R(x\theta\theta', x\theta'') \\ &= R(\rho_2) \cdot h_R(q_{\mathrm{S}}(t), u') \\ &= R(\rho_2) \cdot h_R(q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))), \mathrm{NP}(\mathrm{N}(\mathit{atefl}))) \\ &\quad \cdot h_R(q_{\mathrm{V}}(\mathrm{V}(\mathit{saw})), \mathrm{V}(\mathit{ra'aa})) \\ &\quad \cdot h_R(q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door}))), \mathrm{NP}(\mathrm{N}(\mathit{albab}))) \end{aligned}$$

where we see that the subtrees are evaluated independently and in parallel, whereas the derivation processed the leftmost subtree first. In addition, nondeterminism inside a particular subtree translation is handled locally, whereas nondeterminism is always handled globally in the rewrite semantics.

4 Expressive Power

In this section, we explore the expressive power of xtt and compare the introduced classes of weighted tree transformations. The number of classes was intentionally kept low in order to illustrate a particular approach. A more complete picture is shown in [21], but can also easily be obtained using the techniques recalled here.

Let us first recall the HASSE diagram for the unweighted case of [41, Figure 4.5]. Figure 3 shows the relevant subpart that we are interested in. The contribution [41] contains a much more refined HASSE diagram that relates many more classes. The interested reader might consult [41, Figure 4.5] and translate those additional results to the weighted setting using the approach demonstrated here.

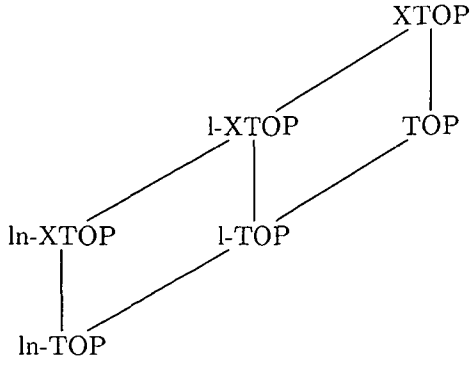


Figure 3: HASSE diagram of the classes of weighted tree transformations computed by xtt.

Theorem 6 (see [41, Theorem 4.11]). *Figure 3 is a* HASSE *diagram if* $(A, +, \cdot, 0, 1)$ *is the* BOOLEAN *semiring.*

The approach that we want to demonstrate only concerns the strictness of the inclusions or the incomparability of classes. Variations of the approach are (implicitly and explicitly) used, for example, in [19, 37, 23]. Since the approach only covers inequalities, the inclusions have to be shown in the standard way. We choose the set of classes of weighted tree transformations such that all inclusions of Figure 3 trivially hold in every semiring. Now let us show how to lift a statement of the form $\mathcal{C} \not\subseteq \mathcal{C}'$ from the BOOLEAN semiring to proper semirings. Recall that a nontrivial semiring is proper if it is not a ring and that every countably complete semiring is proper by Proposition 2.

First we lift the application of a semiring homomorphism $h\colon A \to B$ from semiring elements to weighted tree transformations and to xtt. Given a weighted tree transformation $\tau\colon T_\Sigma \times T_\Delta \to A$, we write $h(\tau)$ for the weighted tree transformation $h(\tau)\colon T_\Sigma \times T_\Delta \to B$ such that $h(\tau)(t,u) = h(\tau(t,u))$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. Moreover, given an xtt $M = (Q, \Sigma, \Delta, I, R)$ we write $h(M)$ for the xtt $h(M) = (Q, \Sigma, \Delta, I, h(R))$ where $h(R)$ is such that $\operatorname{supp}(h(R)) \subseteq \operatorname{supp}(R)$ and $h(R)(\rho) = h(R(\rho))$ for every $\rho \in \operatorname{supp}(R)$.

The next theorem shows that applying an essentially complete semiring homomorphism h to an xtt M yields an xtt $h(M)$ that computes the weighted tree transformation $h(\tau_M)$. In other words, such a homomorphism is also compatible with xtt and its computed weighted tree transformations.

Theorem 7. *Let* $h\colon A \to B$ *be an essentially complete semiring homomorphism. Then* $\tau_{h(M)} = h(\tau_M)$ *for every xtt* M.

Proof. Let $M = (Q, \Sigma, \Delta, I, R)$, $t \in T_\Sigma$, and $u \in T_\Delta$. Then

$$h(\tau_M)(t,u) = h\Bigg(\sum_{\substack{q \in I, k \in \mathbb{N}, \rho_1, \dots, \rho_k \in \operatorname{supp}(R) \\ q(t) \Rightarrow_M^{\rho_1} \cdots \Rightarrow_M^{\rho_k} u}} R(\rho_1) \cdots R(\rho_k)\Bigg)$$

$$
\begin{aligned}
&\overset{\dagger}{=} \sum_{\substack{q\in I,k\in\mathbb{N},\rho_1,\ldots,\rho_k\in\mathrm{supp}(R)\\ q(t)\Rightarrow_M^{\rho_1}\cdots\Rightarrow_M^{\rho_k}u}} h\big(R(\rho_1)\cdots R(\rho_k)\big)\\
&= \sum_{\substack{q\in I,k\in\mathbb{N},\rho_1,\ldots,\rho_k\in\mathrm{supp}(R)\\ q(t)\Rightarrow_M^{\rho_1}\cdots\Rightarrow_M^{\rho_k}u}} h(R(\rho_1))\cdots h(R(\rho_k))\\
&= \sum_{\substack{q\in I,k\in\mathbb{N},\rho_1,\ldots,\rho_k\in\mathrm{supp}(h(R))\\ q(t)\Rightarrow_M^{\rho_1}\cdots\Rightarrow_M^{\rho_k}u}} h(R)(\rho_1)\cdots h(R)(\rho_k)\\
&= \tau_{h(M)}(t,u)\ ,
\end{aligned}
$$

where we used the essential completeness of h in the step marked †. The summands $R(\rho_1)\cdots R(\rho_k)$ in the previous step clearly are in the finitely generated subsemiring $\langle C\rangle$ where $C = \{R(\rho) \mid \rho \in \mathrm{supp}(R)\}$, which is a finite set. □

With the help of the previous theorem we can now prove that if an inclusion is valid in the proper semiring $(A, +, \cdot, 0, 1)$, then it must also be valid in the BOOLEAN semiring. Intuitively, this is achieved by just applying the essentially complete semiring homomorphism h of Section 2. We will typically use this statement as contraposition, if two classes are not contained in the BOOLEAN semiring, then they also are not contained in the proper semiring $(A, +, \cdot, 0, 1)$, which is the desired lift result.

Theorem 8. *Let* $\mathcal{C}, \mathcal{C}' \in \{\text{ln-TOP}, \text{l-TOP}, \text{TOP}, \text{ln-XTOP}, \text{l-XTOP}, \text{XTOP}\}$. *If* $\mathcal{C} \subseteq \mathcal{C}'$ *holds in the proper commutative semiring* $(A, +, \cdot, 0, 1)$, *then it also holds in the* BOOLEAN *semiring.*

Proof. Let h be the essentially complete semiring homomorphism discussed in Section 2. For every xtt N with the properties required by $\mathcal{C}$ over the BOOLEAN semiring, we can easily construct an xtt M (with the same properties) over the proper semiring $(A, +, \cdot, 0, 1)$ such that $h(M) = N$. This can be achieved by reinterpreting N (up to the identity of the unit elements 0 and 1) as an xtt over the semiring $(A, +, \cdot, 0, 1)$. By Theorem 7, we have $h(\tau_M) = \tau_{h(M)} = \tau_N$. Since $\mathcal{C} \subseteq \mathcal{C}'$ is true in the proper semiring $(A, +, \cdot, 0, 1)$, there exists an xtt M' with the properties required by $\mathcal{C}'$ such that $\tau_{M'} = \tau_M$. Note that both M and M' compute over $(A, +, \cdot, 0, 1)$. Again we use Theorem 7 to conclude that the xtt $h(M')$ computes $\tau_{h(M')} = h(\tau_{M'}) = h(\tau_M) = \tau_N$ over the BOOLEAN semiring. It is an easy exercise to verify that $h(M')$ has the same properties (linear, nondeleting, top-down tree transducer) as M'. Thus, we proved that for every xtt over the BOOLEAN semiring with the properties required by $\mathcal{C}$ we can construct an equivalent xtt also over the BOOLEAN semiring with the properties required by $\mathcal{C}'$, which proves the statement. □

Since the inclusions of Figure 3 are trivial and the inequalities can be lifted from the unweighted case using Theorem 8, we can immediately conclude the following theorem.

Theorem 9. *Figure 3 is a* HASSE *diagram for every proper commutative semiring* $(A, +, \cdot, 0, 1)$.

As already indicated the presented method applies just as well to other weighted devices such as weighted string transducers, weighted tree-walking automata, etc. In fact, it would be relatively easy to lift even the full HASSE diagram of [41, Figure 4.5] to the weighted case but since that involves a number of additional notions such as look-ahead, we leave this exercise to the reader.

We end this section with a demonstration of the usefulness of the different semantics and presentations of xtt. Again we do not strive to obtain the most general results, but rather we want to illustrate the principles. We start with *domain* and *range*. Let $\tau\colon T_\Sigma \times T_\Delta \to A$ be a weighted tree transformation. Then the *domain of* τ is the weighted tree language $\varphi\colon T_\Sigma \to A$ such that

$$\varphi(t) = \sum_{u \in T_\Delta} \tau(t, u)$$

for every $t \in T_\Sigma$. Mind that the sum might be infinite. It is finite if for every $t \in T_\Sigma$ there exist only finitely many $u \in T_\Delta$ such that $(t, u) \in \operatorname{supp}(\tau)$. For example, if τ is computed by an xtt without input ε-rules, then this property holds and the sum is finite. Intuitively, if an xtt does not have input ε-rules, then each derivation step consumes at least one input symbol. Thus, the number of derivation steps is limited by $|t|$, which can be used to derive an upper bound for the size of any output tree. If the sum is infinite, then we assume that the semiring $(A, +, \cdot, 0, 1)$ is countably complete with respect to $\sum$ as usual. Dually, the *range of* τ is the weighted tree language $\psi\colon T_\Delta \to A$ such that

$$\psi(u) = \sum_{t \in T_\Sigma} \tau(t, u)$$

for every $u \in T_\Delta$.

To keep the presentation simple, let $\tau = \tau_M$ for some linear and nondeleting xtt without input ε-rules. As already remarked the absence of input ε-rules guarantees that the sum in the definition of the domain is finite. Using our approach the result for arbitrary linear and nondeleting xtt over countably complete semirings can be derived using a result of [35] (see [17, Corollary 6.10]). Here we focus on the domain φ of τ_M. Since M is linear and nondeleting we can use Theorem 3 to obtain an equivalent weighted linear and complete bimorphism $B = (f, \psi, g)$ with $\psi\colon T_\Gamma \to A$. Since $\tau_B = \tau_M$, we can equivalently consider the domain of τ_B. By definition

$$\varphi(t) = \sum_{u \in T_\Delta} \tau_B(t, u) = \sum_{u \in T_\Delta} \Big(\sum_{\substack{s \in T_\Gamma \\ f(s)=t, g(s)=u}} \psi(s) \Big) = \sum_{\substack{s \in T_\Gamma, u \in T_\Delta \\ f(s)=t, g(s)=u}} \psi(s)$$

$$= \sum_{s \in T_\Gamma, f(s)=t} \psi(s) \ ,$$

which can be rewritten as $\varphi = \sum_{s \in T_\Gamma} \psi(s).f(s)$ where

- $\psi(s).f(s)$ is the weighted tree language that is 0 everywhere besides $f(s)$ where the weight is $\psi(s)$, and
- weighted tree languages are added componentwise.

This last presentation shows that the domain is just the application of the linear, complete, and ε-free tree homomorphism f to the recognizable weighted tree language ψ, where ε-free means that $f_k(\gamma) \neq \mathrm{x}_1$ for every $k \in \mathbb{N}$ and $\gamma \in \Gamma$ and it follows from the fact that M has no input ε-rule.

Theorem 10 (see [20, Corollary 8]). *The domain of a linear and nondeleting xtt without input ε-rules is a recognizable weighted tree language.*

Proof. Using the steps presented above and Theorem 2 we immediately obtain the statement. □

Clearly, the range of a linear and nondeleting xtt without output ε-rules is recognizable due to symmetry.

Finally, let us consider the input and the output product, which together with domain and range can be used to prove preservation of recognizability [20, 21]. But let us first define the mentioned input and output product. Given a weighted tree transformation $\tau\colon T_\Sigma \times T_\Delta \to A$ and weighted tree languages $\varphi\colon T_\Sigma \to A$ and $\psi\colon T_\Delta \to A$, the *input product* $\varphi \triangleleft \tau$ of τ by φ and the *output product* $\tau \triangleright \psi$ of τ by ψ are defined by

$$(\varphi \triangleleft \tau)(t,u) = \varphi(t) \cdot \tau(t,u) \qquad \text{and} \qquad (\tau \triangleright \psi)(t,u) = \tau(t,u) \cdot \psi(u) \ ,$$

respectively, for every $t \in T_\Sigma$ and $u \in T_\Delta$.

Often input and output products are handled by specialized Bar-Hillel constructions [5, 45] or compositions [4, 12]. We will discuss the composition approach in the third part of this survey, but let us present an input product construction for weighted tree transformations computed by linear and nondeleting weighted top-down tree transducers and recognizable weighted tree languages. We will show that every such input product can again be computed by a linear and nondeleting weighted top-down tree transducer. A more detailed overview on input and output products can be found in [39].

From now on, let $M = (Q, \Sigma, \Delta, I, R)$ be a linear and nondeleting weighted top-down tree transducer and $N = (P, \Sigma, \delta, F)$ be a wta. We want to construct a linear and nondeleting weighted top-down tree transducer M' such that $\tau_{M'} = \varphi_N \triangleleft \tau_M$. Since M is linear and nondeleting, it visits each input subtree exactly once.

Definition 4. *The input product $N \triangleleft M$ is the weighted top-down tree transducer $(Q', \Sigma, \Delta, I', R')$ where*

- $Q' = Q \times P$,
- $I' = \{(q,p) \mid q \in I, p \in F\}$, *and*
- $R'(l,r) = \delta(p_1 \cdots p_k, \sigma, p) \cdot R(l',r')$ *for all $l \in Q'(T_\Sigma(\mathrm{X}))$ and $r \in T_\Delta(Q'(\mathrm{X}))$ where*
  - *l' and r' are obtained from l and r, respectively, by dropping the second component in the states that occur in l' and r',*

- $p \in P$, $k \in \mathbb{N}$, *and* $\sigma \in \Sigma$ *are such that* $l = (q,p)(\sigma(\mathrm{x}_1, \ldots, \mathrm{x}_k))$ *for some* $q \in Q$, *and*
- *for every* $i \in [k]$, *the state* p_i *is such that there is a position* $w_i \in \mathrm{pos}_{Q'}(r)$ *in the right-hand side with* $r|_{w_i} = (q_i, p_i)(\mathrm{x}_i)$ *for some* $q_i \in Q$.

To illustrate the use of the alternative semantics, we will prove that the input product transducer indeed computes the input product as desired.

Theorem 11 (see [39, Theorem 2]). $\tau_{(N \lhd M)} = \varphi_N \lhd \tau_M$.

Proof. Let $N \lhd M = (Q', \Sigma, \Delta, I', R')$ as in Definition 4. We prove that

$$h_{R'}((q,p)(t), u) = \delta(t,p) \cdot h_R(q(t), u)$$

for every $t \in T_\Sigma$, $u \in T_\Delta$, $q \in Q$, and $p \in P$. Let $\xi = (q,p)(t)$.

$$\begin{aligned}
& h_{R'}(\xi, u) \\
= & \sum_{\substack{l \to r \in \mathrm{supp}(R') \\ (l,\theta') \in \mathrm{match}(\xi) \\ (s,\theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta\colon \mathrm{var}(s) \to Q'(\mathrm{var}(l)) \\ r = s\theta}} R'(l,r) \cdot \prod_{x \in \mathrm{var}(s)} h_{R'}(x\theta\theta', x\theta'') \\
= & \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l,\theta') \in \mathrm{match}(q(t)) \\ (s,\theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta\colon \mathrm{var}(s) \to Q(\mathrm{var}(l)) \\ k = |\mathrm{var}(s)|, p_1, \ldots, p_k \in P \\ r = s\theta}} \delta(p_1 \cdots p_k, l(1), p) \cdot R(l,r) \cdot \prod_{i=1}^{k} h_{R'}(((\mathrm{x}_i\theta)(\varepsilon), p_i)(\mathrm{x}_i\theta'), \mathrm{x}_i\theta'') \\
\overset{\dagger}{=} & \sum_{\substack{l \to r \in \mathrm{supp}(R) \\ (l,\theta') \in \mathrm{match}(q(t)) \\ (s,\theta'') \in \mathrm{match}(u) \\ s \text{ normalized} \\ \theta\colon \mathrm{var}(s) \to Q(\mathrm{var}(l)) \\ k = |\mathrm{var}(s)|, p_1, \ldots, p_k \in P \\ r = s\theta}} \delta(p_1 \cdots p_k, l(1), p) \cdot R(l,r) \cdot \prod_{i=1}^{k} \delta(\mathrm{x}_i\theta', p_i) \cdot h_R(\mathrm{x}_i\theta\theta', \mathrm{x}_i\theta'') \\
\overset{\dagger}{=} & \delta(t,p) \cdot h_R(q(t), u) \ ,
\end{aligned}$$

where we used the induction hypothesis in the step marked †. With the auxiliary statement established, the proof of the main statement is now easy. Let $t \in T_\Sigma$ and $u \in T_\Delta$. Then

$$\begin{aligned}
\tau_{(N \lhd M)}(t,u) &= \sum_{q' \in I'} h_{R'}(q'(t), u) = \sum_{q \in I, p \in F} h_{R'}((q,p)(t), u) \\
&= \sum_{q \in I, p \in F} \delta(t,p) \cdot h_R(q(t), u) = \varphi_N(t) \cdot \tau_M(t,u) = (\varphi_N \lhd \tau_M)(t,u) \ .
\end{aligned}$$

□

Acknowledgments

The author would like to express his gratitude to the reviewers. Their insight and remarks improved the article. In addition, the author would like to acknowledge the financial support of the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

References

[1] Alexandrakis, Athanasios and Bozapalidis, Symeon. Weighted grammars and Kleene's theorem. *Inf. Process. Lett.*, 24(1):1–4, 1987.

[2] Arnold, André and Dauchet, Max. Bi-transductions de forêts. In *Proc. 3rd Int. Coll. Automata, Languages and Programming*, pages 74–86. Edinburgh University Press, 1976.

[3] Arnold, André and Dauchet, Max. Morphismes et bimorphismes d'arbres. *Theoret. Comput. Sci.*, 20(4):33–93, 1982.

[4] Baker, Brenda S. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41(2):186–213, 1979.

[5] Bar-Hillel, Yehoshua, Perles, Micha, , and Shamir, Eliyahu. On formal properties of simple phrase structure grammars. In Bar-Hillel, Yehoshua, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison Wesley, 1964.

[6] Berstel, Jean and Reutenauer, Christophe. Recognizable formal power series on trees. *Theoret. Comput. Sci.*, 18(2):115–148, 1982.

[7] Courcelle, Bruno and Franchi–Zannettacci, Paul. Attribute grammars and recursive program schemes. *Theor. Comput. Sci.*, 17(1):163–191 & 235–257, 1982.

[8] Dauchet, Max. *Transductions inversibles de forêts.* Thèse 3ème cycle, Université de Lille, 1975.

[9] Droste, Manfred, Kuich, Werner, and Vogler, Heiko, editors. *Handbook of Weighted Automata.* EATCS Monographs on Theoret. Comput. Sci. Springer, 2009.

[10] Droste, Manfred, Stüber, Torsten, and Vogler, Heiko. Weighted finite automata over strong bimonoids. *Inf. Sci.*, 180(1):156–166, 2010.

[11] Eilenberg, Samuel. *Volume A: Automata, Languages, and Machines*, volume 59 of *Pure and Applied Math.* Academic Press, 1974.

[12] Engelfriet, Joost. Bottom-up and top-down tree transformations: A comparison. *Math. Systems Theory*, 9(3):198–231, 1975.

[13] Engelfriet, Joost. Top-down tree transducers with regular look-ahead. *Math. Systems Theory*, 10(1):289–303, 1976.

[14] Engelfriet, Joost, Fülöp, Zoltán, and Vogler, Heiko. Bottom-up and top-down tree series transformations. *J. Autom. Lang. Combin.*, 7(1):11–70, 2002.

[15] Engelfriet, Joost and Vogler, Heiko. Macro tree transducers. *J. Comput. System Sci.*, 31(1):71–146, 1985.

[16] Engelfriet, Joost and Vogler, Heiko. Modular tree transducers. *Theor. Comput. Sci.*, 78(2):267–303, 1991.

[17] Ésik, Zoltán and Kuich, Werner. Formal tree series. *J. Autom. Lang. Combin.*, 8(2):219–285, 2003.

[18] Fülöp, Zoltán. On attributed tree transducers. *Acta Cybernet.*, 5(1):261–279, 1981.

[19] Fülöp, Zoltán, Gazdag, Zsolt, and Vogler, Heiko. Hierarchies of tree series transformations. *Theoret. Comput. Sci.*, 314(3):387–429, 2004.

[20] Fülöp, Zoltán, Maletti, Andreas, and Vogler, Heiko. Preservation of recognizability for synchronous tree substitution grammars. In *Proc. 1st Workshop Applications of Tree Automata in Natural Language Processing*, pages 1–9. Association for Computational Linguistics, 2010.

[21] Fülöp, Zoltán, Maletti, Andreas, and Vogler, Heiko. Weighted extended tree transducers. submitted, 2011.

[22] Fülöp, Zoltán and Vogler, Heiko. Weighted tree transducers. *J. Autom. Lang. Combin.*, 9(1):31–54, 2004.

[23] Fülöp, Zoltán and Vogler, Heiko. Weighted tree automata and tree transducers. In Droste et al. [9], chapter 9, pages 313–403.

[24] Gécseg, Ferenc and Steinby, Magnus. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[25] Gécseg, Ferenc and Steinby, Magnus. Tree languages. In Rozenberg, Grzegorz and Salomaa, Arto, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.

[26] Golan, Jonathan S. *Semirings and their Applications.* Kluwer Academic, Dordrecht, 1999.

[27] Graehl, Jonathan and Knight, Kevin. Training tree transducers. In *Proc. 2004 Human Language Technology Conf. NAACL*, pages 105–112. Association for Computational Linguistics, 2004.

[28] Hebisch, Udo and Weinert, Hanns J. *Semirings — Algebraic Theory and Applications in Computer Science.* World Scientific, 1998.

[29] Hopcroft, John E. and Ullman, Jeffrey D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[30] Jurafsky, Daniel and Martin, James H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Processing.* Prentice-Hall, 2000.

[31] Karner, Georg. Continuous monoids and semirings. *Theoret. Comput. Sci.*, 318(3):355–372, 2004.

[32] Knight, Kevin. Capturing practical natural language transformations. *Machine Translation*, 21(2):121–133, 2007.

[33] Knight, Kevin and Graehl, Jonathan. An overview of probabilistic tree transducers for natural language processing. In *Proc. 6th Int. Conf. Intelligent Text Processing and Computational Linguistics*, pages 1–24. Association for Computational Linguistics, 2005.

[34] Kuich, Werner. Formal power series over trees. In *Proc. 3rd Int. Conf. Developments in Language Theory*, pages 61–101. Aristotle University of Thessaloniki, 1998.

[35] Kuich, Werner. Tree transducers and formal tree series. *Acta Cybernet.*, 14(1):135–149, 1999.

[36] Kuich, Werner and Salomaa, Arto. *Semirings, Automata, Languages*, volume 5 of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer, 1986.

[37] Maletti, Andreas. The power of tree series transducers of type I and II. In *Proc. 9th Int. Conf. Developments in Language Theory*, volume 3572 of LNCS, pages 338–349. Springer, 2005.

[38] Maletti, Andreas. Compositions of extended top-down tree transducers. *Inform. and Comput.*, 206(9–10):1187–1196, 2008.

[39] Maletti, Andreas. Input and output products for weighted extended top-down tree transducers. In *Proc. 14th Int. Conf. Developments in Language Theory*, volume 6224 of LNCS, pages 316–327. Springer, 2010.

[40] Maletti, Andreas. Why synchronous tree substitution grammars? In *Proc. Human Language Technology Conf. NAACL*, pages 876–884. Association for Computational Linguistics, 2010.

[41] Maletti, Andreas, Graehl, Jonathan, Hopkins, Mark, and Knight, Kevin. The power of extended top-down tree transducers. *SIAM J. Comput.*, 39(2):410–430, 2009.

[42] Manning, Chris and Schütze, Hinrich. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[43] Raoult, Jean-Claude. Rational tree relations. *Bull. Belg. Math. Soc.*, 4(1):149–176, 1997.

[44] Rounds, William C. Mappings and grammars on trees. *Math. Syst. Theory*, 4(3):257–287, 1970.

[45] Satta, Giorgio. Translation algorithms by means of language intersection. *Manuscript*, 2011. available at: `http://www.dei.unipd.it/~satta`.

[46] Shieber, Stuart M. Synchronous grammars as tree transducers. In *Proc. 7th Int. Workshop Tree Adjoining Grammar and Related Formalisms*, pages 88–95, 2004.

[47] Thatcher, James W. Generalized^2 sequential machine maps. *J. Comput. System Sci.*, 4(4):339–367, 1970.

[48] Thatcher, James W. Tree automata: an informal survey. In Aho, Alfred V., editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall, 1973.

[49] Wang, Huaxiong. On characters of semirings. *Houston J. Math.*, 23(3):391–405, 1997.

[50] Wang, Huaxiong. On rational series and rational languages. *Theoret. Comput. Sci.*, 205(1–2):329–336, 1998.

Regular Papers

Acta Cybernetica **20** (2011) 253–267.

Classes of Tree Languages and DR Tree Languages Given by Classes of Semigroups

Ferenc Gécseg*

Abstract

In the first section of the paper we give general conditions under which a class of recognizable tree languages with a given property can be defined by a class of monoids or semigroups defining the class of string languages having the same property. In the second part similar questions are studied for classes of (DR) tree languages recognized by deterministic root-to-frontier tree recognizers.

Keywords: recognizable tree languages, DR recognizable tree languages, syntactic semigroups, syntactic monoids

1 Introduction

In [3] we characterized the class of recognizable monotone string languages and that of recognizable monotone tree languages by means of syntactic monoids. It turned out that both classes can be defined by the class **M** of monoids whose right unit submonoids are closed under divisors, i.e. a recognizable string or tree language is monotone if and only if its syntactic monoid is in **M**. This was the observation which motivated the writing of paper [1], where such characterizations from more general classes of string languages have been lifted to classes of (frontier-to-root) tree languages.

In [4] we obtained results for the classes of definite and nilpotent deterministic root-to-frontier (DR) tree languages similar to those in [3]. The aim of this paper is to strengthen the main result of [1], on one hand, and to give general conditions under which a class of DR tree languages with a given property can be defined by a class of monoids or semigroups defining the class of string languages having the same property, on the other hand. The proofs are based on the observation that the syntactic monoids (syntactic semigroups) of recognizable tree languages and the syntactic path monoids (syntactic path semigroups) of DR tree languages can be given as subdirect products of the syntactic monoids (syntactic semigroups)

*Department of Informatics, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary

of suitable recognizable string languages. We shall show for the classes of DR-monotone, DR-nilpotent and DR-definite tree languages that they satisfy these conditions.

It should be noted that the classes of tree languages considered in this paper are not necessarily varieties. For readers interested in varieties of recognizable tree languages, we refer to the fundamental papers [11] and [13].

2 Notions and Notation

Sets of operational symbols will be denoted by Σ. If Σ is finite and nonvoid, then it is called a *ranked alphabet.* For the subset of Σ consisting of all m-ary operational symbols from Σ we shall use the notation Σ_m $(m \geq 0)$. By a Σ-*algebra* we mean a pair $\mathcal{A} = (A, \{\sigma^{\mathcal{A}} | \sigma \in \Sigma\})$, where $\sigma^{\mathcal{A}}$ is an m-ary operation on A if $\sigma \in \Sigma_m$. If there will be no danger of confusion then we omit the superscript $\mathcal{A}$ in $\sigma^{\mathcal{A}}$ and simply write $\mathcal{A} = (A, \Sigma)$. Finally, all algebras considered in this paper will be finite, i.e. A is finite and Σ is a ranked alphabet.

Take a Σ-algebra $\mathcal{A} = (A, \Sigma)$, a $\sigma \in \Sigma_m$ $(m > 0)$, an i $(1 \leq i \leq m)$ and $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m \in A$. Then $\sigma(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_m)$ is an *elementary translation symbol* of $\mathcal{A}$. The set of all elementary translation symbols of $\mathcal{A}$ will be denoted by $\mathrm{ETS}(\mathcal{A})$. In the sequel elementary translation symbols will be considered as unary operational symbols. Moreover, $\mathrm{ETalg}(\mathcal{A})$ will denote the unary algebra $(A, \mathrm{ETS}(\mathcal{A}))$ with

$$\sigma(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_m)^{\mathrm{ETalg}(\mathcal{A})}(a) = \sigma^{\mathcal{A}}(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_m)$$
$$(\sigma(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_m) \in \mathrm{ETS}(\mathcal{A}),\ a \in A).$$

Let X be a set of variables. The set $T_\Sigma(X)$ of ΣX-*trees* (or Σ-*trees over* X) is defined as follows:

(i) $X \subseteq T_\Sigma(X)$,

(ii) $\sigma(p_1, \ldots, p_m) \in T_\Sigma(X)$ if $m \geq 0$, $\sigma \in \Sigma_m$ and $p_1, \ldots, p_m \in T_\Sigma(X)$, and

(iii) every ΣX-tree can be obtained by applying the rules (i) and (ii) a finite number of times.

In the sequel X will stand for the countable set $\{x_1, x_2, \ldots\}$, and for every $n \geq 0$, X_n will denote the subset $\{x_1, \ldots, x_n\}$ of X. A subset of $T_\Sigma(X_n)$ is called a ΣX_n-*language.* If Σ or X_n is not specified then we speak of a *tree language.*

Take a Σ-algebra $\mathcal{A} = (A, \Sigma)$ and a tree $p \in T_\Sigma(X_n)$. Let us define the mapping $p^{\mathcal{A}} : A^n \to A$ in the following way: for any $\mathbf{a} = (a_1, \ldots, a_n) \in A^n$,

(i) if $p = x_i \in X_n$, then $p^{\mathcal{A}}(\mathbf{a}) = a_i$,

(ii) if $p = \sigma(p_1, \ldots, p_m)$ $(\sigma \in \Sigma_m,\ p_1, \ldots, p_m \in T_\Sigma(X_n))$, then

$$p^{\mathcal{A}}(\mathbf{a}) = \sigma^{\mathcal{A}}(p_1^{\mathcal{A}}(\mathbf{a}), \ldots, p_m^{\mathcal{A}}(\mathbf{a})).$$

If there is no danger of confusion, then we omit $\mathcal{A}$ in $p^{\mathcal{A}}$.

A ΣX_n-*recognizer* is a system $\mathbf{A} = (\mathcal{A}, \mathbf{a}, A')$, where

(i) $\mathcal{A} = (A, \Sigma)$ is an algebra,

(ii) $\mathbf{a} = (a^{(1)}, \ldots, a^{(n)})$ $(a^{(1)}, \ldots, a^{(n)} \in A)$ is the *initial* vector,

(iii) $A' \subseteq A$ is the set of *final* states.

If $n = 1$, then we usually write $a^{(1)}$ for $(a^{(1)})$. Moreover, it is said that $\mathbf{A}$ is *connected* if $\{p(\mathbf{a}) \mid p \in T_\Sigma(X_n)\} = A$.

If Σ and X_n are not specified then we speak of a *tree recognizer*. Furthermore, if $\Sigma = \Sigma_1$ and $n = 1$, then $\mathbf{A}$ is a *finite state recognizer*, shortly *recognizer*. If we are dealing with recognizers, then (unary) trees are sometimes written as words: for a tree $\sigma_1(\ldots(\sigma_k(x_1))\ldots)$ we may write $\sigma_k \ldots \sigma_1$.

The *tree language* $T(\mathbf{A})$ *recognized* by the ΣX_n-recognizer $\mathbf{A} = (\mathcal{A}, \mathbf{a}, A')$ is given by

$$T(\mathbf{A}) = \{p \in T_\Sigma(X_n) \mid p(\mathbf{a}) \in A'\}.$$

The class of recognizable tree languages will be denoted by **Treelang**, and **Lang** is its subclass consisting of all tree languages recognizable by finite state recognizers.

Let **Prop** be a property of recognizable tree languages. The best way is to define **Prop** as a subclass of **Treelang**. If **K** is a subclass of **Treelang**, then **Prop(K)** will denote the class of all tree languages which are simultaneously in **Prop** and **K**.

If not otherwise specified, $\mathbf{A}$ will be the ΣX_n-recognizer $(\mathcal{A}, \mathbf{a}, A')$. Here $\mathcal{A}$ is a Σ-algebra (A, Σ), $\mathbf{a} = (a^{(1)}, \ldots, a^{(n)})$ and $A' \subseteq A$. Consider a ΣX_n-recognizer $\mathbf{A}$. For each $x \in X_n \cup \Sigma_0$, define the finite state $\mathrm{ETS}(\mathcal{A})$-recognizer $\mathbf{A}_x = (\mathcal{A}_x, a_x, A'_x)$ in the following way:

(1) $a_x = \begin{cases} a^{(i)}, & \text{if } x = x_i \ (1 \leq i \leq n), \\ \sigma^{\mathcal{A}}, & \text{if } x = \sigma \in \Sigma_0. \end{cases}$

(2) $A_x = \{p^{\mathrm{ETalg}(\mathcal{A})}(a_x) \mid p \in T_{\mathrm{ETS}(\mathcal{A})}(X_1)\}$.

(3) $\mathcal{A}_x = (A_x, \mathrm{ETS}(\mathcal{A}))$ is a subalgebra of $\mathrm{ETalg}(\mathcal{A})$.

(4) $A'_x = A_x \cap A'$.

These $\mathbf{A}_x$ are called *translation recognizers* of $\mathbf{A}$.

Let $\hat{T}_\Sigma(X_n)$ denote the set of all Σ-trees over $X_n \cup \{*\}$ $(* \notin X_n)$ in which $*$ occurs exactly once. Elements in $\hat{T}_\Sigma(X_n)$ are *special trees* of Thomas [14] and Heuter [9]. Let us define the product $q \cdot p$ of $q \in T_\Sigma(X_n) \cup \hat{T}_\Sigma(X_n)$ and $p \in \hat{T}_\Sigma(X_n)$ by $q \cdot p = p(q)$. (Here and in the sequel, for any $p \in \hat{T}_\Sigma(X_n)$ and $q \in T_\Sigma(X_n) \cup \hat{T}_\Sigma(X_n)$, $p(q)$ is obtained by replacing the occurrence of $*$ in p by q $(p(q) = p(* \leftarrow q))$.) Obviously, under this multiplication $\hat{T}_\Sigma(X_n)$ is a monoid with the identity element $*$.

Let $T \subseteq T_\Sigma(X_n)$ be a tree language. Define the binary relation μ_T on $\hat{T}_\Sigma(X_n)$ in the following way: for any $p, q \in \hat{T}_\Sigma(X_n)$,

$$p \equiv q(\mu_T) \Longleftrightarrow$$

$$(\forall p', p'' \in \hat{T}_\Sigma(X_n),\ x \in X_n \cup \Sigma_0)((p' \cdot p \cdot p'')(x) \in T \Longleftrightarrow (p' \cdot q \cdot p'')(x) \in T).$$

This μ_T is a congruence of the monoid $\hat{T}_\Sigma(X_n)$, which is called the *syntactic congruence* of T. Moreover, the quotient monoid $\hat{T}_\Sigma(X_n)/\mu_T$ is the *syntactic monoid* of T, which will be denoted by $\mathrm{Syntm}(T)$.

The restriction of μ_T to $\hat{T}_\Sigma(X_n) \setminus \{*\}$ will be denoted by the same μ_T. The quotient semigroup $\hat{T}_\Sigma(X_n) \setminus \{*\}/\mu_T$ is the *syntactic semigroup* of T. The syntactic semigroup of T will be denoted by $\mathrm{Synts}(T)$.

We say that a property **Prop** of recognizable tree languages *can be defined by a class* $\mathbf{M}$ *of monoids,* if for all $T \in$ **Treelang**, $T \in$ **Prop** $\Longleftrightarrow \mathrm{Syntm}(T) \in \mathbf{M}$. Similarly, a property **Prop** of recognizable tree languages *can be defined by a class* $\mathbf{S}$ *of semigroups,* if for all $T \in$ **Treelang**, $T \in$ **Prop** $\Longleftrightarrow \mathrm{Synts}(T) \in \mathbf{S}$. For any ΣX_n-recognizer $\mathbf{A}$ and $p \in \hat{T}_\Sigma(X_n)$, let $p(\mathbf{a})$ stand for $p(x_1 \leftarrow a^{(1)}, \ldots, x_n \leftarrow a^{(n)})$ and $p(\mathbf{a})(a)$ for $p(\mathbf{a})(* \leftarrow a)$ $(a \in A)$, i.e. $p(\mathbf{a})(a)$ is obtained from p by replacing the occurrences of x_i by $a^{(i)}$ and that of $*$ by a.

Let Y be an ordinary alphabet, Y^* the free semigroup generated by Y and $L \subseteq Y^*$ a language over Y. Furthermore, let μ_L be the binary relation on Y^* given by $u \equiv v(\mu_L)$ $(u, v \in Y^*)$ iff for any $u', u'' \in Y^*$ the equivalence $u'uu'' \in L \Longleftrightarrow u'vu'' \in L$ holds. As it is well known, μ_L is a congruence relation on the free monoid Y^*, and the quotient monoid Y^*/μ_L is called the *syntactic monoid* of L. Let the same μ_L denote the restriction of μ_L to the semigroup $Y^+ = Y^* \setminus \{e\}$, where e is the empty word. The quotient semigroup Y^+/μ_L is the *syntactic semigroup* of L. It is obvious, if finite state recognizers are taken as special tree recognizers, then the above two definitions of syntactic monoids coincide. The same is true for syntactic semigroups.

For notions and notation not defined in this paper, see [6] and [7].

3 Tree languages

Let $\mathbf{A}$ be an arbitrary connected ΣX_n-recognizer. Define the mapping $\epsilon_\mathbf{A} : \hat{T}_\Sigma(X_n) \to T_{\mathrm{ETS}(A)}(*)$ in the following way:

1) $\epsilon_\mathbf{A}(*) = *$.

2) If $p = \sigma(p_1, \ldots, p_{i-1}, p_i, p_{i+1}, \ldots, p_m)$ $(\sigma \in \Sigma_m,\ p_j \in T_\Sigma(X_n),\ j \in \{1, \ldots, i-1, i+1, \ldots, m\},\ p_i \in \hat{T}_\Sigma(X_n))$, then

$$\epsilon_\mathbf{A}(p) = \sigma(p_1^\mathcal{A}(\mathbf{a}), \ldots, p_{i-1}^\mathcal{A}(\mathbf{a}), \epsilon_\mathbf{A}(p_i), p_{i+1}^\mathcal{A}(\mathbf{a}), \ldots, p_m^\mathcal{A}(\mathbf{a})).$$

Since $\mathbf{A}$ is connected, $\epsilon_\mathbf{A}$ is an onto mapping. If there is no danger of confusion we shall omit $\mathbf{A}$ in $\epsilon_\mathbf{A}$.

Let $T \subseteq T_\Sigma(X_n)$ be a tree language. For each $x \in X_n \cup \Sigma_0$, define the binary relation $\mu_{T,x}$ on $\hat{T}_\Sigma(X_n)$ in the following way: for any $p, q \in \hat{T}_\Sigma(X_n)$,

$$p \equiv q(\mu_{T,x}) \Longleftrightarrow$$

$$(\forall p', p'' \in \hat{T}_\Sigma(X_n))((p' \cdot p \cdot p'')(x) \in T \Longleftrightarrow (p' \cdot q \cdot p'')(x) \in T).$$

Clearly, these relations $\mu_{T,x}$ are congruences of the monoid $\hat{T}_\Sigma(X_n)$.

By the definitions of the syntactic monoid and the syntactic semigroup of a ΣX_n-language T we obviously have the following two results.

Lemma 1. *The syntactic monoid* Syntm(T) *is isomorphic to a subdirect product of the monoids* $\hat{T}_\Sigma(X_n)/\mu_{T,x}$, $x \in X_n \cup \Sigma_0$. ◇

Lemma 2. *The syntactic semigroup* Synts(T) *is isomorphic to a subdirect product of the semigroups* $\hat{T}_\Sigma(X_n) \setminus \{*\}/\mu_{T,x}$, $x \in X_n \cup \Sigma_0$, *where the restriction of* $\mu_{T,x}$ *to* $\hat{T}_\Sigma(X_n) \setminus \{*\}$ *is denoted by the same* $\mu_{T,x}$. ◇

We now show

Lemma 3. *Let* $\mathbf{A}$ *be an arbitrary connected* ΣX_n*-recognizer. Then for all* $x \in X_n \cup \Sigma_0$,

$$\hat{T}_\Sigma(X_n)/\mu_{T,x} \cong \text{Syntm}(\text{T}(\mathbf{A}_\text{x})).$$

Proof. It is obvious that for any two $p, q \in \hat{T}_\Sigma(X_n)$ we have $\epsilon(p \cdot q) = \epsilon(p) \cdot \epsilon(q)$. We show that for all $p, q \in \hat{T}_\Sigma(X_n)$,

$$p \equiv q(\mu_{T,x}) \Longleftrightarrow \epsilon(p) \equiv \epsilon(q)(\mu_{T(\mathbf{A}_x)}).$$

Remember that ϵ is an onto mapping since $\mathbf{A}$ is connected. Thus,

$$\begin{array}{c}
p \equiv q(\mu_{T,x}) \\
\Updownarrow \\
(\forall r, s \in \hat{T}_\Sigma(X_n))((r \cdot p \cdot s)(x) \in T \Longleftrightarrow (r \cdot q \cdot s)(x) \in T) \\
\Updownarrow \\
(\forall r, s \in \hat{T}_\Sigma(X_n))((r \cdot p \cdot s)(\mathbf{a})(a_x) \in A' \Longleftrightarrow (r \cdot q \cdot s)(\mathbf{a})(a_x) \in A') \\
\Updownarrow \\
(\forall r, s \in \hat{T}_\Sigma(X_n))(\epsilon(r \cdot p \cdot s)(a_x) \in A'_x \Longleftrightarrow \epsilon(r \cdot q \cdot s)(a_x) \in A'_x) \\
\Updownarrow \\
(\forall r, s \in \hat{T}_\Sigma(X_n))((\epsilon(r) \cdot \epsilon(p) \cdot \epsilon(s)) \in A'_x \Longleftrightarrow (\epsilon(r) \cdot \epsilon(q) \cdot \epsilon(s)) \in A'_x) \\
\Updownarrow \\
\epsilon(p) \equiv \epsilon(q)(\mu_{T(\mathbf{A}_x)}).
\end{array}$$

Therefore, $p/\mu_{T,x} \to \epsilon(p)/\mu_{T(\mathbf{A}_x)}$ $(p \in \hat{T}_\Sigma(X_n))$ is an isomorphic mapping of $\hat{T}_\Sigma(X_n)/\mu_{T,x}$ onto $\text{Syntm}(\text{T}(\mathbf{A}_\text{x}))$. ◇

The following lemma can be proved in a similar way.

Lemma 4. *Let* $\mathbf{A}$ *be an arbitrary connected* ΣX_n*-recognizer. Then for all* $x \in X_n \cup \Sigma_0$,

$$\hat{T}_\Sigma(X_n) \setminus \{*\}/\mu_{T,x} \cong \mathrm{Synts}(\mathrm{T}(\mathbf{A}_x)).$$

◇

Let $\mathbf{S}$ be a class of semigroups. We say that $\mathbf{S}$ is *closed under subdirect products,* if all subdirect products of semigroups from $\mathbf{S}$ with finitely many factors are in $\mathbf{S}$. Moreover, $\mathbf{S}$ is *closed under subdirect factors,* if whenever a subdirect product of two semigroups is in $\mathbf{S}$, then both of them are in $\mathbf{S}$.

In this paper all classes of semigroups will contain only finite semigrouups.

We are now ready to state and prove

Theorem 1. *Let* **Prop** *be a property of recognizable tree languages. Assume that the following conditions are satisfied:*

(1) *For every* ΣX_n*-language* T *there exists a connected* ΣX_n*-recognizer* $\mathbf{A}$ *with* $T(\mathbf{A}) = T$ *such that*

$$T \in \mathbf{Prop} \Longleftrightarrow (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) $\mathbf{Prop(Lang)}$ *can be defined by a class* $\mathbf{M}$ *of monoids.*

(3) $\mathbf{M}$ *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* $\mathbf{M}$.

Proof. Assume that the conditions of our theorem are satisfied.

First take a $T \in \mathbf{Treelang}$ with $\mathrm{Syntm}(T) \in \mathbf{M}$, and let $\mathbf{A}$ be a connected ΣX_n-recognizer such that $T = T(\mathbf{A})$ satisfies (1). By Lemma 1 and 3, $\mathrm{Syntm}(T)$ is isomorphic to a subdirect product of the monoids $\mathrm{Syntm}(T(\mathbf{A}_x))$ $(x \in X_n \cup \Sigma_0)$. From this, by (3), we obtain that $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$, and thus, by (2), $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$ for all $x \in X_n \cup \Sigma_0$, which, by (1), implies that $T = T(\mathbf{A}) \in \mathbf{Prop}$.

Conversely, assume that $T \in \mathbf{Prop}$, and let $\mathbf{A}$ be a connected tree recognizer with $T = T(\mathbf{A})$ satisfying (1). Then, for each $x \in X_n \cup \Sigma_0$, $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$. Thus, by (2), $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$. Again, by Lemma 1 and 3, $\mathrm{Syntm}(T)$ is isomorphic to a subdirect product of $\mathrm{Syntm}(T(\mathbf{A}_x))$ $(x \in X_n \cup \Sigma_0)$. Moreover, by (3), $\mathbf{M}$ is closed under subdirect products. Therefore, $\mathrm{Synt}(T) \in \mathbf{M}$. ◇

The next theorem can be proved in a similar way.

Theorem 2. *Let* **Prop** *be a property of recognizable tree languages. Assume that the following conditions are satisfied:*

(1) *For every* ΣX_n*-language* T *there exists a connected* ΣX_n*-recognizer* $\mathbf{A}$ *with* $T(\mathbf{A}) = T$ *such that*

$$T \in \mathbf{Prop} \Longleftrightarrow (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **S** *of semigroups.*

(3) **S** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* **S**. ◇

In [1] we proved

Theorem 3. *Let* **Prop** *be a property of recognizable tree languages. Assume that the following conditions are satisfied.*

(1) *For all minimal tree recognizers* **A**,

$$T(\mathbf{A}) \in \mathbf{Prop} \Longleftrightarrow (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **M** *of monoids.*

(3) **M** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* **M**. ◇

It is easy to show that the previous theorem is true for properties defined by semigroups:

Theorem 4. *Let* **Prop** *be a property of recognizable tree languages. Assume that the following conditions are satisfied.*

(1) *For all minimal tree recognizers* **A**,

$$T(\mathbf{A}) \in \mathbf{Prop} \Longleftrightarrow (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **S** *of semigroups.*

(3) **S** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* **S**. ◇

We shall need

Lemma 5. *If* **A** *and* **B** *are equivalent connected* ΣX_n*-recognizers then for all* $x \in X_n \cup \Sigma_0$, $\mathrm{Syntm}(T(\mathbf{A}_x)) \cong \mathrm{Syntm}(T(\mathbf{B}_x))$.

Proof. For a $p \in \hat{T}_\Sigma(X_n)$ set $\overline{p} = \epsilon_{\mathbf{A}}(p)$ and $\overline{\overline{p}} = \epsilon_{\mathbf{B}}(p)$. Let $p, q \in \hat{T}_\Sigma(X_n)$ and $x \in X_n$ be arbitrary. We have

$$\begin{array}{c}
(\forall r,s \in \hat{T}_\Sigma(X_n))(((r\cdot p\cdot s)(x))^{\mathcal{A}}(\mathbf{a}) \in A' \Leftrightarrow ((r\cdot q\cdot s)(x))^{\mathcal{A}}(\mathbf{a}) \in A') \overset{T(\mathbf{A})=T(\mathbf{B})}{\Longleftrightarrow} \\
(\forall r,s \in \hat{T}_\Sigma(X_n))(((r\cdot p\cdot s)(x))^{\mathcal{B}}(\mathbf{b}) \in B' \Leftrightarrow ((r\cdot q\cdot s)(x))^{\mathcal{B}}(\mathbf{b}) \in B') \\
\Updownarrow \\
(\forall r,s \in \hat{T}_\Sigma(X_n))\overline{r\cdot p\cdot s}^{\mathcal{A}_x}(a_x) \in A' \Leftrightarrow \overline{r\cdot q\cdot s}^{\mathcal{A}_x}(a_x) \in A') \Longleftrightarrow \\
(\forall r,s \in \hat{T}_\Sigma(X_n))\overline{\overline{r\cdot p\cdot s}}^{\mathcal{B}_x}(b_x) \in B' \Leftrightarrow \overline{\overline{r\cdot q\cdot s}}^{\mathcal{B}_x}(b_x) \in B') \\
\Updownarrow \\
(\forall \overline{r},\overline{s} \in \hat{T}_\Sigma(X_n))((\overline{r}\cdot\overline{p}\cdot\overline{s})^{\mathcal{A}_x}(a_x) \in A' \Leftrightarrow (\overline{r}\cdot\overline{q}\cdot\overline{s})^{\mathcal{A}_x}(a_x) \in A') \Longleftrightarrow \\
(\forall \overline{\overline{r}},\overline{\overline{s}} \in \hat{T}_\Sigma(X_n))((\overline{\overline{r}}\cdot\overline{\overline{p}}\cdot\overline{\overline{s}})^{\mathcal{B}_x}(b_x) \in B' \Leftrightarrow (\overline{\overline{r}}\cdot\overline{\overline{q}}\cdot\overline{\overline{s}})^{\mathcal{B}_x}(b_x) \in B') \\
\Updownarrow \\
\overline{p} \equiv \overline{q}(\mu_{\mathbf{A}_x}) \Leftrightarrow \overline{\overline{p}} \equiv \overline{\overline{q}}(\mu_{\mathbf{B}_x}).
\end{array}$$

Therefore the mapping $\overline{p}/\mu_{\mathbf{A}_x} \to \overline{\overline{p}}/\mu_{\mathbf{B}_x}$ $(p \in \hat{T}_\Sigma(X_n))$ is an isomorphism between the monoids $\mathrm{Syntm}(T(\mathbf{A}_x))$ and $\mathrm{Syntm}(T(\mathbf{B}_x))$. ◇

The following result can be proved in a similar way.

Lemma 6. *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are equivalent connected* ΣX_n*-recognizers then for all* $x \in X_n \cup \Sigma_0$, $\mathrm{Synts}(T(\mathbf{A}_x)) \cong \mathrm{Synts}(T(\mathbf{B}_x))$. ◇

Now we show

Theorem 5. *Theorems 1 and 3 are equivalent.*

Proof. It is obvious that Theorem 1 implies Theorem 3.

To prove the opposite direction, suppose that Theorem 3 is valid and the conditions of Theorem 1 are satisfied. Take a recognizable ΣX_n-tree T and let $\mathbf{A}$ be the minimal ΣX_n-recognizer for T.

First assume that $T \in \mathbf{Prop}$. Let $\mathbf{B}$ be a connected ΣX_n-recognizer recognizing T such that $T(\mathbf{B}_x) \in \mathbf{Prop(Lang)}$ for all $x \in \hat{T}_\Sigma(X_n)$. Therefore, by (2) in Theorem 1, $\mathrm{Syntm}(T(\mathbf{B}_x)) \in \mathbf{M}$. By Lemma 5, $\mathrm{Syntm}(T(\mathbf{A}_x)) \cong \mathrm{Syntm}(T(\mathbf{B}_x))$, thus $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$, which by (2) in Theorem 1 implies $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$.

Conversely, suppose that $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$ for all $x \in \hat{T}_\Sigma(X_n)$. By (2) in Theorem 1, $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$. Let $\mathbf{B}$ be a connected ΣX_n-recognizer with $T(\mathbf{B}) = T(\mathbf{A})$ which satisfies (1) in Theorem 1. By Lemma 5, $\mathrm{Syntm}(T(\mathbf{A}_x))$ and $\mathrm{Syntm}(T(\mathbf{B}_x))$ are isomorphic. Then $\mathrm{Syntm}(T(\mathbf{B}_x)) \in \mathbf{M}$. Therefore, by (2) in Theorem 1, $T(\mathbf{B}_x) \in \mathbf{Prop(Lang)}$. From this, using (1) in Theorem 1, we obtain that $T(\mathbf{A}) = T(\mathbf{B}) \in \mathbf{Prop}$.

We have obtained that the conditions of Theorem 3 are also satisfied. Therefore, **Prop** can be defined by $\mathbf{M}$. ◇

Using a similar proof, one can show

Theorem 6. *Theorems 2 and 4 are equivalent.* ◇

We now show that Theorem 3 is equivalent to

Theorem 7. *Let* **Prop** *be a property of recognizable tree languages and* $\mathbf{M}$ *a class of monoids. Assume that the following conditions are satisfied:*

(1) *For every* ΣX_n*-language* T *and all connected* ΣX_n*-recognizers* $\mathbf{A}$ *with* $T(\mathbf{A}) = T$ *we have*

$$T \in \mathbf{Prop} \iff (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) $\mathbf{Prop(Lang)}$ *can be defined by* $\mathbf{M}$.

(3) $\mathbf{M}$ *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* $\mathbf{M}$.

Proof. It is obvious that Theorem 3 implies Theorem 7.

The opposite direction can be shown by the same idea as the second part of the proof of Theorem 5. Suppose that Theorem 7 is valid and the conditions of Theorem 3 are satisfied. Take a recognizable ΣX_n-tree T.

First assume that $T \in \mathbf{Prop}$. Let $\mathbf{B}$ be a connected ΣX_n-recognizer recognizing T. Moreover, let $\mathbf{A}$ be the minimal ΣX_n-recognizer for T. By our assumption, $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$ for all $x \in \hat{T}_\Sigma(X_n)$. Then, by (2) in Theorem 3, $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$. By Lemma 5, $\mathrm{Syntm}(T(\mathbf{A}_x)) \cong \mathrm{Syntm}(T(\mathbf{B}_x))$, thus $\mathrm{Syntm}(T(\mathbf{B}_x)) \in \mathbf{M}$, which by (2) in Theorem 3 implies $T(\mathbf{B}_x) \in \mathbf{Prop(Lang)}$.

Conversely, let $\mathbf{B}$ be a connected ΣX_n-recognizer with $T(\mathbf{B}) = T$ such that $T(\mathbf{B}_x) \in \mathbf{Prop(Lang)}$ for all $x \in \hat{T}_\Sigma(X_n)$. By condition (2) in Theorem 3, $\mathrm{Syntm}(T(\mathbf{B}_x)) \in \mathbf{M}$, and thus $\mathrm{Syntm}(T(\mathbf{A}_x)) \in \mathbf{M}$ since $\mathrm{Syntm}(T(\mathbf{A}_x))$ and $\mathrm{Syntm}(T(\mathbf{B}_x))$ are isomorphic. Therefore, again by (2) in Theorem 3, $T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}$. From this, using (1) in Theorem 3, we obtain that $T(\mathbf{B}) = T(\mathbf{A}) \in \mathbf{Prop}$.

We have obtained that the conditions of Theorem 7 are satisfied. Therefore, **Prop** can be defined by $\mathbf{M}$. ◇

Summarizing our equivalence results, we have

Theorem 8. *Theorems 1, 3 and 7 are equivalent.* ◇

Using the same technique as in the proof of Theorem 5, one can show that Theorems 2 and 4 are equivalent to

Theorem 9. *Let* **Prop** *be a property of recognizable tree languages. Assume that the following conditions are satisfied:*

(1) *For every ΣX_n-language T and all connected ΣX_n-recognizers $\mathbf{A}$ with $T = T(\mathbf{A})$ we have*

$$T \in \mathbf{Prop} \iff (\forall x \in X_n \cup \Sigma_0)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* $\mathbf{S}$ *of semigroups.*

(3) $\mathbf{S}$ *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be defined by* $\mathbf{S}$.

4 DR tree languages

First of all, we recall several well known concepts from the theory of root-to-frontier tree recognizers.

In what follows, the frequently recurring phrase *deterministic root-to-frontier* is usually abbreviated directly to DR. As before, Σ is a ranked alphabet and X is a (nonempty) frontier alphabet. As usual, in this section we shall suppose that $\Sigma_0 = \emptyset$.

In the study of DR tree languages, which form a proper subclass of all recognizable tree languages, a natural counterpart of syntactic semigroups are syntactic path semigroups introduced in [8]. Thus, for defining classes of DR recognizable tree languages we shall use path semigroups. We have also changed the definition of properties of tree languages defined by tree automata (monotonicity, nilpotency etc) in such a way which is more natural for DR recognizers. To distinguish them from the general definition, we shall use the prefix DR.

A finite *DR Σ-algebra* consists of a non-empty finite set A and a Σ-indexed family of *root-to-frontier operations*

$$\sigma^{\mathcal{A}} : A \longrightarrow A^m \quad (\sigma \in \Sigma_m).$$

Again we write simply $\mathcal{A} = (A, \Sigma)$. A *DR ΣX_n-recognizer* is now defined as a system $\mathbf{A} = (\mathcal{A}, a_0, \mathbf{a})$, where $\mathcal{A} = (A, \Sigma)$ is a finite DR Σ-algebra, $a_0 \in A$ is the *initial state*, and $\mathbf{a} = (A^{(1)}, \ldots, A^{(n)}) \in (\wp A)^n$ is the *final state vector*. ($\wp A$ denotes the power-set of a set A.)

To define the tree language recognized by $\mathbf{A}$, we introduce a mapping $\alpha_{\mathbf{A}}$ of $T_\Sigma(X_n)$ into $\wp A$:

(1) $\alpha_{\mathbf{A}}(x_i) = A^{(i)}$ for $x_i \in X_n$,

(2) $\alpha_{\mathbf{A}}(p) = \{a \in A \mid \sigma^{\mathcal{A}}(a) \in \alpha_{\mathbf{A}}(p_1) \times \ldots \times \alpha_{\mathbf{A}}(p_m)\}$ for $p = \sigma(p_1, \ldots, p_m)$ $(\sigma \in \Sigma_m,\ p_1, \ldots, p_m \in T_\Sigma(X_n))$.

The tree language *recognized* by $\mathbf{A}$ is now defined as the set

$$T(\mathbf{A}) = \{p \in T_\Sigma(X_n) \mid a_0 \in \alpha_{\mathbf{A}}(p)\}.$$

A ΣX_n-tree language is *DR recognizable* if it is recognized by some DR ΣX_n-recognizer. Such tree languages are called *DR tree languages*.

All DR Σ-algebras considered in this paper are supposed to be finite.

Set $\hat{\Sigma} = \bigcup(\{\sigma_1, \ldots, \sigma_m\} \mid \sigma \in \Sigma_m,\ m > 0)$. For any $x \in X_n$ the set $g_x(p) \subseteq \hat{\Sigma}^*$ of *x-paths* in a given ΣX_n-tree p is defined as follows:

(1) $g_x(x) = e$,

(2) $g_x(y) = \emptyset$ for $y \in X_n$, $y \neq x$,

(3) $g_x(p) = \sigma_1 g_x(p_1) \cup \ldots \cup \sigma_m g_x(p_m)$ for $p = \sigma(p_1, \ldots, p_m)$.

For $T \subseteq T_\Sigma(X_n)$ and $x \in X_n$, set $T_x = \bigcup(g_x(p) \mid p \in T)$.

For a DR ΣX_n-recognizer $\mathbf{A} = (\mathcal{A}, a_0, \mathbf{a})$ and $x \in X_n$, now define the recognizer $\mathbf{A}_x = (\hat{\Sigma}, A, a_0, \delta, A^{(i)})$ by $\delta(a, \sigma_j) = \pi_j(\sigma(a))$ $(a \in A,\ \sigma \in \Sigma)$, where $x = x_i$ and π_j is the jth projection of a vector. (Since $\mathbf{A}_x$ are used to recognize words (paths) they are written in the standard form of finite state recognizers.)

We shall use the following obvious result.

Lemma 7. *For all DR recognizable ΣX_n-tree language T and $x \in X_n$, T_x is a recognizable language.* ◇

The *syntactic path congruence* of a ΣX_n-tree language T is the relation on $\hat{\Sigma}^*$ defined by the following condition. For any $w_1, w_2 \in \hat{\Sigma}^*$,

$$w_1 \,\hat{\mu}_T\, w_2 \iff (\forall x \in X)(\forall u, v \in \hat{\Sigma}^*)(uw_1v \in T_x \iff uw_2v \in T_x).$$

The *syntactic path monoid* $\mathrm{Synpm}(T)$ of T is $\hat{\Sigma}^*/\hat{\mu}_T$. Denote by the same $\hat{\mu}_T$ the restriction of $\hat{\mu}_T$ to $\hat{\Sigma}^+$. Then $\hat{\Sigma}^+/\hat{\mu}_T$ is called the *syntactic path semigroup* of T and it is denoted by $\mathrm{Synps}(T)$.

The following facts are obvious since in both cases $\hat{\mu}_T$ is the intersection of the usual syntactic congruences of the languages T_x $(x \in X)$.

Lemma 8. *For any DR ΣX_n-tree language T, $\hat{\Sigma}^*/\hat{\mu}_T$ is isomorphic to a subdirect product of the syntactic monoids $\hat{\Sigma}^*/\hat{\mu}_{T_x}$ $(x \in X_n)$. Similarly, $\hat{\Sigma}^+/\hat{\mu}_T$ is isomorphic to a subdirect product of the syntactic semigroups $\hat{\Sigma}^+/\hat{\mu}_{T_x}$ $(x \in X_n)$.* ◇

A *DR property* is a class of DR tree languages. We say that a DR property **Prop** *can be path-defined by a class* **M** *of monoids,* if for all DR tree languages T, $T \in \mathbf{Prop} \iff \mathrm{Syntpm}(T) \in \mathbf{M}$. Moreover, a DR-property **Prop** *can be path-defined by a class* **S** *of semigroups,* if for all DR tree languages T, $T \in \mathbf{Prop} \iff \mathrm{Syntps}(T) \in \mathbf{S}$.

Using Lemma 8, the next result can be proved in the same way as Theorem 1.

Theorem 10. *Let* **Prop** *be a DR property. Assume that the following conditions are satisfied:*

(1) *For every DR ΣX_n-language T there exists a DR ΣX_n-recognizer* **A** *with $T(\mathbf{A}) = T$ such that*

$$T \in \mathbf{Prop} \iff (\forall x \in X_n)(T(\mathbf{A}_x) \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **M** *of monoids.*

(3) **M** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be path-defined by* **M**. ◇

By the proof of Lemma 7, the above result can be formulated as follows.

Theorem 11. *Let* **Prop** *be a DR property. Assume that the following conditions are satisfied:*

(1) *For every DR ΣX_n-language T,*

$$T \in \mathbf{Prop} \iff (\forall x \in X_n)(T_x \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **M** *of monoids.*

(3) **M** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be path-defined by* **M**. ◇

One can also show

Theorem 12. *Let* **Prop** *be a DR property. Assume that the following conditions are satisfied:*

(1) *For every DR* ΣX_n*-language* T,

$$T \in \mathbf{Prop} \iff (\forall x \in X_n)(T_x \in \mathbf{Prop(Lang)}).$$

(2) **Prop(Lang)** *can be defined by a class* **S** *of semigroups.*

(3) **S** *is closed under subdirect products and subdirect factors.*

Then **Prop** *can be path-defined by* **S**.

4.1 DR monotone trée languages

It is said that a DR ΣX_n-recognizer $\mathbf{A} = (A, a_0, \mathbf{a})$ is *DR monotone* if there exists a partial ordering $\leq$ on A such that $\pi_i(\sigma(a)) \geq a$ for all $\sigma \in \Sigma_m$, $1 \leq i \leq m$ and $a \in A$. Moreover, a tree language $T \subseteq T_\Sigma(X_n)$ is *DR monotone*, if $T = T(\mathbf{A})$ for a DR monotone ΣX_n-recognizer $\mathbf{A}$.

Let S be a semigroup and $s \in S$ an arbitrary element. It is said that $r \in S$ is a *divisor* of s if $s = rt$ or $s = tr$ for some $t \in S$. A subsemigroup S' of S is *closed under divisors* if S' contains all divisors of each of its elements. Moreover, we say that a subsemigroup S' of S is a *right-unit subsemigroup* if there exists an $s \in S$ such that $S' = \{r \in S \mid s = sr\}$. More precisely, in this case S' is called the right-unit subsemigroup of S *belonging* to s.

The class of monoids whose all right-unit subsemigroups are closed under divisors will be denoted by $\mathbf{M}_{\text{cld}}$.

The following result from [3] gives a semigroup-theoretic characterization of monotone languages.

Theorem 13. *A recognizable language* L *is monotone iff every right-unit subsemigroup of the syntactic monoid of* L *is closed under divisors.* ◇

Thus the class of monotone languages is defined by the class $\mathbf{M}_{\text{cld}}$ of monoids. The next result is from [1].

Theorem 14. *The class of all monotone tree languages together with* $\mathbf{M}_{\text{cld}}$ *satisfies the conditions of Theorem 3.* ◇

For DR monotone tree languages we have

Theorem 15. *The class* **Prop** *of all DR monotone tree languages with* $\mathbf{M} = \mathbf{M}_{\text{cld}}$ *satisfies the conditions of Theorem 11.*

Proof. It has been shown in [1] that (2) and (3) are true.

For showing (1), take a DR monotone ΣX_n-language T. For all $x \in X_n$, T_x are monotone. Therefore, $T_x \in \mathbf{Prop(Lang)}$.

Conversely, assume that for all $x_i \in X_n$, T_{x_i} are monotone. Therefore, there are monotone recognizers $\mathbf{B}_i = (\hat{\Sigma}, B_i, b_{i_0}, \delta_i, B_i')$ with partial orderings $\leq_i$ on B_i such that $T(\mathbf{B}_i) = T_{x_i}$. Define the DR ΣX_n-recognizer $\mathbf{B} = (\mathcal{B}, b_0, \mathbf{b})$ in the following way:

(1) $\mathcal{B} = (B, \Sigma)$ where $B = B_1 \times \ldots \times B_n$ and for all $(b_1, \ldots, b_n) \in B$ and $\sigma \in \Sigma_m$,

$$\sigma^{\mathcal{B}}(b_1, \ldots, b_n) = \\ ((\delta_1(b_1, \sigma_1), \ldots, \delta_n(b_n, \sigma_1)), \ldots, (\delta_1(b_1, \sigma_m), \ldots, \delta_n(b_n, \sigma_m))).$$

(2) $b_0 = (b_{1_0} \ldots, b_{n_0})$.

(3) $B^{(i)} = B_1 \times \ldots \times B_{i-1} \times B_i{}' \times B_{i+1} \times \ldots \times B_n$.

It is a routine work to show that $T(\mathbf{B}) = T$. Define the relation $\leq$ on B by

$$((b_1, \ldots, b_n) \leq (b_1', \ldots, b_n')) \Longleftrightarrow ((\forall i \in \{1, \ldots, n\})(b_i \leq_i b_i'))$$

$$((b_1, \ldots, b_n), (b_1', \ldots, b_n') \in B).$$

Easy to show that $\leq$ is a partial ordering and $(b_1, \ldots, b_n) \leq \pi_j(\sigma(b_1, \ldots, b_n))$ for all $(b_1, \ldots, b_n) \in B$, $\sigma \in \Sigma$ and $j \in \{1, \ldots, n\}$. Thus, $T \in \mathbf{Prop}$. $\diamondsuit$

Since the class of DR tree languages is a proper subclass of the class of all tree languages both the class of all monotone tree languages and the class of DR monotone tree languages can be defined by the same class $\mathbf{M}_{\mathrm{cld}}$ of monoids, one could come to the hypothesis that the class of all DR monotone tree languages is the restriction of the class of all monotone tree languages to the class of DR tree languages. However, in [3] it was shown that the class of DR *monotone* tree languages and that of monotone tree languages are incomparable.

4.2 DR nilpotent tree languages

Let $\mathcal{A} = (A, \Sigma)$ be a DR Σ-algebra, $a \in A$ an element and $p \in T_\Sigma(X_n)$ a tree. Define the word $\overline{\mathrm{fr}}(ap) \in A^*$ in the following way:

1) if $p = x \in X_n$, then $\overline{\mathrm{fr}}(ap) = a$,

2) if $p = \sigma(p_1, \ldots, p_m)$ and $(a_1, \ldots, a_m) = \sigma^{\mathcal{A}}(a)$, then

$$\overline{\mathrm{fr}}(ap) = \overline{\mathrm{fr}}(a_1 p_1) \ldots \overline{\mathrm{fr}}(a_m p_m).$$

A DR ΣX_n-algebra $\mathcal{A} = (A, \Sigma)$ is *DR nilpotent* if there are an integer $k \geq 0$ and an element $\bar{a} \in A$ such that for all $a \in A$ and $p \in T_\Sigma(X_n)$ with $\mathrm{mh}(p) \geq k$, $\overline{\mathrm{fr}}(ap) = \bar{a}^l$ for a natural number l. ($\bar{a}$ is called the *nilpotent element* of $\mathcal{A}$ and $\mathrm{mh}(p)$

is the length of the shortest path of p.) A DR ΣX_n-recognizer $\mathbf{A} = (\mathcal{A}, a_0, \mathbf{a})$ is *DR nilpotent* if $\mathcal{A}$ is DR nilpotent. Moreover, a ΣX_n-tree language T is *DR nilpotent* if it can be recognized by a DR nilpotent ΣX_n-recognizer.

A semigroup S is nilpotent if it has a zero-element 0 and there is a non-negative integer k such that $s_1 \dots s_k = 0$ for all $s_1, \dots, s_k \in S$.

The class of all nilpotent semigroups will be denoted by $\mathbf{S}_{\text{nil}}$.

The following result from [12] gives a semigroup-theoretic characterization of nilpotent languages. (See, also [5].)

Theorem 16. *A recognizable language L is nilpotent iff the syntactic semigroup of L is nilpotent.* ◇

Thus the class of nilpotent languages can be defined by the class $\mathbf{S}_{\text{nil}}$ of semigroups.

Theorem 17. *The class of all DR nilpotent tree languages with $\mathbf{S} = \mathbf{S}_{\text{nil}}$ satisfy the conditions of Theorem 12.*

Proof. It has been proved in [1] that (2) and (3) are true.

Condition (1) can be shown in a similar way as (1) in the proof of Theorem 15 by replacing "DR monotone" with "DR nilpotent", taking $(b_1, \dots, b_k)$ to be the nilpotent element if b_i is the nilpotent element of $\mathbf{B}_i$, and disregarding the partial ordering.

4.3 DR definite tree languages

Let S be a semigroup. It is said that S is *right regular* if the equality $ss_I = s_I$ holds in S for any element s and idempotent s_I. The class of all right regular semigroups will be denoted by $\mathbf{S}_{\text{rr}}$

Let $k \geq 0$ be an arbitrary integer. A DR Σ-algebra $\mathcal{A} = (A, \Sigma)$ is *DR k-definite* if $\overline{\text{fr}}(ap) = \overline{\text{fr}}(a'p)$ for all $a, a' \in A$ and $p \in T_\Sigma(X_n)$ with $\text{mh}(p) \geq k$. A DR ΣX_n-recognizer $\mathbf{A} = (\mathcal{A}, a_0, \mathbf{a})$ is *DR k-definite* if $\mathcal{A}$ is DR k-definite. Moreover, a ΣX_n-tree language T is *DR k-definite* if it can be recognized by a DR k-definite ΣX_n-recognizer. Finally, T is *DR definite* if it is DR k-definite for some k.

It is well known that the class of all definite languages can be defined by the class of all right regular semigroups. Thus, condition (2) of Theorem 12 is satisfied by $\mathbf{S}_{\text{rr}}$. We now show

Theorem 18. *The class of all DR definite tree languages with $\mathbf{S} = \mathbf{S}_{\text{rr}}$ satisfies the conditions of Theorem 12.*

Proof. Condition (3) of Theorem 12 is obviously satisfied by $\mathbf{S}_{\text{rr}}$.

It is obvious that if $T \subseteq T_\Sigma(X_n)$ is DR k-definite then so are T_x for all $x \in X_n$. Conversely, assume that T_{x_i} are k_i-definite. There are k_i-definite recognizers $\mathbf{B}_i = (\hat{\Sigma}, B_i, b_{i_0}, \delta_i, B_i')$ such that $T(\mathbf{B}_i) = T_{x_i}$. Again take the DR ΣX_n-recognizer $\mathbf{B} = (\mathcal{B}, b_0, \mathbf{b})$ obtained by the construction used in the proof of Theorem 15. Let $k = \max(k_i \mid i = 1, \dots, n)$. It is easy to show that $\mathbf{B}$ is k-definite and $T(\mathbf{B}) = T$.

References

[1] Gécseg, F. Classes of tree languages determined by classes of monoids. *International Journal of Foundations of Computer Science,* **18** (2007), 1237-1246.

[2] Gécseg, F. and Imreh, B. On a special class of tree automata. In *2nd Conf. on Automata, Languages and Programming Systems* (Salgótarján, 1988), 141-152.

[3] Gécseg, F. and Imreh, B. On momotone automata and monotone languages. *Journal of Automata, Languages and Combinatorics,* **7** (2002), 71-82.

[4] Gécseg, F. and Imreh, B. On definite and nilpotent DR tree languages. *Journal of Automata, Languages and Combinatorics,* **9** (2004), 55-60.

[5] Gécseg, F. and Peák, I. *Algebraic theory of automata.* (Akadémiai Kiadó, Budapest, 1972).

[6] Gécseg, F. and Steinby, M. *Tree automata.* (Akadémiai Kiadó, Budapest, 1984).

[7] Gécseg, F. and Steinby, M. Tree languages. In *Handbook of Formal Languages, Vol. 3,* eds. G. Rozenberg and A. Salomaa (Springer-Verlag, Berlin, 1997), 1-68.

[8] Gécseg, F. and Steinby, M. Minimal Recognizers and Syntactic Monoids of DR Tree Languages. In Words, Semigroups, & Transductions, World Scientifics, (2001), 155-167.

[9] Heuter, U. Definite tree languages. *Bulletin of the EATCS,* **35** (1988), 137-142.

[10] Pin, J.-E. Syntactic semigroups. In *Handbook of Formal Languages, Vol. 1,* eds. G. Rozenberg and A. Salomaa (Springer-Verlag, Berlin, 1997), 679-746.

[11] Salehi, S. Varieties of tree languages definable by syntactic monoids. *Acta Cybernetica,* **17** (2005), 21-41.

[12] Ševrin, L. N. On some classes of abstract automata (Russian). *Uspehi matem. nauk,* **17:6(108)** (1962), 219.

[13] Steinby, M. A theory of tree language varieties. In *Tree Automata and Languages,* eds. M. Nivat and A. Podelski (Elsevier, Amsterdam, 1992), 5781.

[14] Thomas, W. Logical aspects in the study of tree languages. In *Ninth Colloquium on Trees in Algebra and in Programming* (Cambridge University Press, Cambridge 1984), 270-280.

Received 2nd November 2010

Acta Cybernetica **20** (2011) 269–283.

Cooperating Distributed Grammar Systems with Random Context Grammars as Components

Zbyněk Křivka* and Tomáš Masopust†

Abstract

In this paper, we discuss cooperating distributed grammar systems where components are (variants of) random context grammars. We give an overview of known results and open problems, and prove some further results.

Keywords: Cooperating distributed grammar system, random context grammar, left-random context grammar.

1 Introduction

Rewriting systems based on a simple form of productions play an important role in formal language theory. Therefore, it is no surprise that context-free grammars and their variants are frequently studied models. However, many systems describing real-life applications, such as parsers of natural and programming languages, require some additional mechanisms that allow to check for context dependencies. From that viewpoint, context-free grammars are not fully convenient for those applications because they are too simple to handle such dependencies.

A natural method of handling more context dependencies with rewriting systems is to compose systems of several components, and to define a cooperation protocol for these components to generate a common sentential form. Such devices are known as *cooperating distributed (CD) grammar systems* [2, 3, 12]. Components are represented by grammars or other rewriting devices, and the protocol for mutual cooperation describes (roughly speaking) the number of steps one component has to perform before allowing another component to work. For instance, the most interesting protocol is the so-called *terminal derivation mode* (*t*-mode, for short) making the component work until it is not able to perform another derivation step. It is well-known that the cooperation has a significant effect on context-free grammars. Namely, working in non-trivial modes, context-free CD grammar systems are more powerful than ordinary context-free grammars [3].

*Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic. E-mail: krivka@fit.vutbr.cz

†Institute of Mathematics of the Czech Academy of Sciences, Žižkova 22, 616 62 Brno, Czech Republic. E-mail: masopust@ipm.cz

Thus, rewriting systems that are simple and able to check for context dependencies are of interest as components of CD grammar systems [10]. One of such systems are *random context grammars* [14], which are a natural generalization of context-free grammars with respect to context dependency checking. Specifically, in random context grammars, two finite sets of non-terminals are attached to each context-free production—a *permitting* and a *forbidding* set—and such a production is applicable only if all permitting symbols appear in the current sentential form, while no forbidding symbol does. The family of random context languages contains properly the family of context-free languages and is properly included in the family of context-sensitive languages (coincides with the family of recursively enumerable languages, respectively, if erasing productions are allowed [1, 14]). In addition, random context grammars with all permitting (forbidding) sets empty result in the introduction of *forbidding (permitting) grammars*, which are less powerful than random context grammars (the reader is referred to [7, 15], respectively, for more details and for some pumping-like properties of those languages).

In this paper, we discuss the generative power of several variants of CD grammar systems with random context grammars as components, give an overview of known results and open problems, and prove some further results.

2 Preliminaries and Definitions

We assume that the reader is familiar with formal language theory [6, 12, 13]. An alphabet is a finite non-empty set. For an alphabet V, V^* represents the free monoid generated by V. The unit of V^*, the empty string, is denoted by λ, and the free semigroup generated by V is denoted by $V^+ = V^* - \{\lambda\}$. For a string $w \in V^*$, let $|w|$ denote the length of w and $alph(w)$ denote the set of all symbols occurring in w. Let **CF**, **CS**, and **RE** denote the families of context-free, context-sensitive, and recursively enumerable languages, respectively.

A *random context grammar* [14] is a quadruple $G = (N, T, P, S)$, where N is the alphabet of non-terminals, T is the alphabet of terminals such that $N \cap T = \emptyset$, $V = N \cup T$, $S \in N$ is the start symbol, and P is a finite set of productions of the form $(A \to x, Per, For)$, where $A \to x$ is a context-free production, $A \in N$ and $x \in V^*$, and $Per, For \subseteq N$. For $u, v \in V^*$ and a production $(A \to x, Per, For) \in P$, the relation $uAv \Rightarrow uxv$ holds provided that

$$Per \subseteq alph(uv) \quad \text{and} \quad alph(uv) \cap For = \emptyset. \tag{1}$$

The transitive closure and the reflexive and transitive closure of $\Rightarrow$ are denoted by $\Rightarrow^+$ and $\Rightarrow^*$, respectively. The language generated by G is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$. A *permitting (forbidding) grammar* is a random context grammar $G = (N, T, P, S)$, where for each production $(A \to x, Per, For) \in P$, it holds that $For = \emptyset$ ($Per = \emptyset$, respectively). The language families generated by random context grammars, permitting grammars, and forbidding grammars are denoted by $\mathbf{RC}_\lambda$, $\mathbf{PER}_\lambda$, and $\mathbf{FOR}_\lambda$, respectively, and by **RC**, **PER**, and **FOR**, respectively, if they are generated by corresponding grammars without erasing productions.

A *left-random context grammar* [4, 8] is a quadruple $G = (N, T, P, S)$, where N, T, P, and S are the same as in random context grammars. For $u, v \in V^*$ and a production $(A \to x, Per, For) \in P$, we define the relation $uAv \Rightarrow uxv$ provided that $Per \subseteq alph(u)$

and $alph(u) \cap For = \emptyset$. That is, only the symbols on the left side of the rewritten non-terminal are considered. The language generated by G is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$. A *left-permitting (left-forbidding) grammar* is a left-random context grammar $G = (N, T, P, S)$, where for each production $(A \to x, Per, For) \in P$, it holds that $For = \emptyset$ ($Per = \emptyset$, respectively). The language families generated by left-random context grammars, left-permitting grammars, and left-forbidding grammars are denoted by $\ell\mathbf{RC}_\lambda$, $\ell\mathbf{PER}_\lambda$, and $\ell\mathbf{FOR}_\lambda$, respectively, and by $\ell\mathbf{RC}$, $\ell\mathbf{PER}$, and $\ell\mathbf{FOR}$, respectively, if they are generated by grammars without erasing productions.

2.1 Cooperating Distributed Grammar Systems

A *cooperating distributed (CD) grammar system* is a construct $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, $n \geq 1$, where N is the alphabet of non-terminals, T is the alphabet of terminals, $N \cap T = \emptyset$, $S \in N$ is the start symbol, and for $1 \leq i \leq n$, each component P_i is a finite set of context-free productions. For $u, v \in V^*$, $V = N \cup T$, and $1 \leq k \leq n$, let $u \Rightarrow_k v$ denote a derivation step performed by the application of a production from P_k. As usual, extend the relation $\Rightarrow_k$ to $\Rightarrow_k^m$ (the m-step derivation), $m \geq 0$, $\Rightarrow_k^+$, and $\Rightarrow_k^*$. In addition, we define the relation $u \Rightarrow_k^t v$ so that $u \Rightarrow_k^+ v$ and there is no $w \in V^*$ such that $v \Rightarrow_k w$. The languages generated by Γ working in the f-mode, $f \in \{*, t\} \cup \{\leq m, =m, \geq m : m \geq 1\}$, denoted by $L_f(\Gamma)$, is defined as follows.

t-mode $L_t(\Gamma) = \{w \in T^* :$ there are $\ell \geq 1$ and sentential forms $\alpha_i, 1 \leq i \leq \ell$, such that $\alpha_i \Rightarrow_{k_i}^t \alpha_{i+1}, 1 \leq k_i \leq n, \alpha_1 = S$, and $\alpha_\ell = w\}$.

$*$-mode $L_*(\Gamma) = \{w \in T^* :$ there are $\ell \geq 1$ and sentential forms $\alpha_i, 1 \leq i \leq \ell$, such that $\alpha_i \Rightarrow_{k_i}^* \alpha_{i+1}, 1 \leq k_i \leq n, \alpha_1 = S$, and $\alpha_\ell = w\}$.

$=m$-mode $L_{=m}(\Gamma) = \{w \in T^* :$ there are $\ell \geq 1$ and sentential forms $\alpha_i, 1 \leq i \leq \ell$, such that $\alpha_i \Rightarrow_{k_i}^m \alpha_{i+1}, 1 \leq k_i \leq n, \alpha_1 = S$, and $\alpha_\ell = w\}, m \geq 1$.

$\leq m$-mode $L_{\leq m}(\Gamma) = \{w \in T^* :$ there are $\ell \geq 1$ and sentential forms $\alpha_i, 1 \leq i \leq \ell$, such that $\alpha_i \Rightarrow_{k_i}^{j_i} \alpha_{i+1}, 1 \leq k_i \leq n, 1 \leq j_i \leq m, \alpha_1 = S$, and $\alpha_\ell = w\}$.

$\geq m$-mode $L_{\geq m}(\Gamma) = \{w \in T^* :$ there are $\ell \geq 1$ and sentential forms $\alpha_i, 1 \leq i \leq \ell$, such that $\alpha_i \Rightarrow_{k_i}^{j_i} \alpha_{i+1}, 1 \leq k_i \leq n, 1 \leq m \leq j_i, \alpha_1 = S$, and $\alpha_\ell = w\}$.

Language families generated by CD grammar systems with n context-free components working in the f-mode are denoted by $\mathbf{CD}_f(\mathbf{CF}_\lambda, n)$, or $\mathbf{CD}_f(\mathbf{CF}, n)$ if the components are non-erasing. The following results are well-known [3].

1. $\mathbf{CD}_f(\mathbf{CF}_\lambda, n) = \mathbf{CD}_f(\mathbf{CF}, n) = \mathbf{CF}$, for $n \geq 1$ and $f \in \{=1, \geq 1, *\} \cup \{\leq k : k \geq 1\}$,
2. $\mathbf{CF} \subset \mathbf{CD}_f(\mathbf{CF}, 2) \subseteq \mathbf{CD}_f(\mathbf{CF}, r) \subseteq \mathbf{MAT}$ and
   $\mathbf{CF} \subset \mathbf{CD}_f(\mathbf{CF}_\lambda, 2) \subseteq \mathbf{CD}_f(\mathbf{CF}_\lambda, r) \subseteq \mathbf{MAT}_\lambda$, for $f \in \{=k, \geq k : k \geq 2\}$ and $r \geq 3$,
3. $\mathbf{CD}_t(\mathbf{CF}_\lambda, 2) = \mathbf{CD}_t(\mathbf{CF}, 2) = \mathbf{CF}$ and $\mathbf{CD}_t(\mathbf{CF}_\lambda, n) = \mathbf{CD}_t(\mathbf{CF}, n) = \mathbf{ET0L}$, for $n \geq 3$,

where **ET0L** denotes the family of languages generated by extended tabled interaction-less Lindenmayer systems [12], and **MAT** and $\mathbf{MAT}_\lambda$ denote the families of languages generated by matrix grammars without and with erasing productions [6], respectively.

Obviously, the definition of CD grammar systems can be generalized so that the components are sets of productions of any type. This leads to several new definitions of CD grammar systems with (permitting, forbidding, left-permitting, left-forbidding, left-) random context grammars as components. The family of languages generated by CD grammar systems with n components of type $\mathbf{X}$ working in the f-mode, $f \in \{*,t\} \cup \{\leq k, =k, \geq k : k \geq 1\}$, is denoted by $\mathbf{CD}_f(\mathbf{X}_\lambda, n)$, or $\mathbf{CD}_f(\mathbf{X}, n)$ if the components are non-erasing.

3 Results

First, let us recall that $\mathbf{CF} \subset \mathbf{PER}_\lambda = \mathbf{PER} \subset \mathbf{RC} \subset \mathbf{CS}$, $\mathbf{CF} \subset \mathbf{FOR}_\lambda \subset \mathbf{RC}_\lambda = \mathbf{RE}$, and $\mathbf{CF} \subset \mathbf{FOR} \subset \mathbf{RC}$ [7, 15, 16]. In addition, in the case of left-forbidding grammars $\ell\mathbf{FOR}_\lambda = \ell\mathbf{FOR} = \mathbf{CF}$ [8], i.e., left-forbidding languages coincide with context-free languages. However, it is of interest to compare this with results concerning context-free CD grammar systems and left-forbidding CD grammar systems (below), where it turns out that although the components are of the same power, left-forbidding CD grammar systems are more powerful than context-free CD grammar systems. Furthermore, in the case of left-permitting grammars $\mathbf{CF} \subset \ell\mathbf{PER}$ [4], which is surprising in comparison with the result concerning left-forbidding grammars. The inclusion is clear from the definition, and the strictness follows from the following example [4].

Example 1. Let $G = (\{S,A,C,A',C'\}, \{a,b,c\}, P, S)$ be a left-permitting grammar, where

$$P = \{(S \to AC, \emptyset), (A \to aA'b, \emptyset), (A \to ab, \emptyset), (A' \to A, \emptyset), \\ (C \to cC', \{A'\}), (C \to c, \emptyset), (C' \to C, \{A\})\}.$$

We can see that $L(G) = \{a^n b^n c^m : n \geq m \geq 1\}$, which is a non-context-free language.

Note that not all the relations among language families **PER**, **FOR**, ℓ**PER**, **RC**, ℓ**RC**, **CS** (and analogously among their erasing variants) are known. More specifically, only the relations depicted in Figure 1 are known. The reader is also referred to [1] for more details.

Open problem 1. *What are the relations among the above mentioned language families?*

3.1 Alternative Definition of the Direct Derivation Step

In the case of random context grammars, the direct derivation step is in the literature also defined so that the rewritten symbol is considered by the context-dependency checking mechanism (cf. [14] and [11]), i.e., for $u, v \in V^*$ and a production $(A \to x, Per, For) \in P$, the relation $uAv \Rightarrow uxv$ holds provided that

$$Per \subseteq alph(uAv) \quad \text{and} \quad alph(uAv) \cap For = \emptyset. \tag{2}$$

Lemma 1. *Definitions* (1) *and* (2) *are equivalent for random context (permitting, forbidding) grammars.*

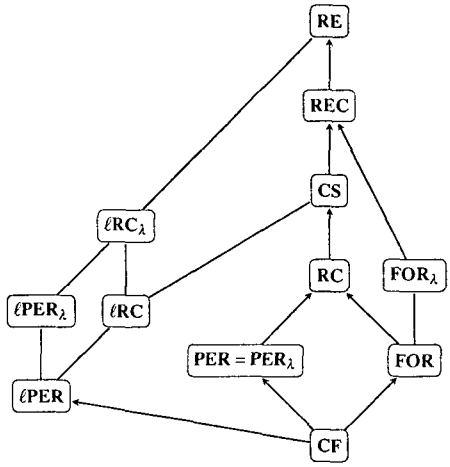


Figure 1: A hierarchy of language families. If two families are connected by a line (an arrow), the upper family includes (includes properly) the lower family. If two families are not connected, they are not necessarily incomparable.

Proof. (1) $\Rightarrow$ (2): Let $G = (N, T, P, S)$ be a random context (permitting, forbidding) grammar using definition (1). Construct the grammar $G' = (N \cup N', T, P', S)$ of the same type using definition (2) so that $N' = \{A' : A \in N\}$, $N \cap N' = \emptyset$, and P' is defined as follows.

(a) $P' = \{(A \to A', \emptyset, N'), (A' \to x, Per, For) : (A \to x, Per, For) \in P\}$ for forbidding and random context grammars.

(b) $P' = \{(A \to A', \emptyset, \emptyset), (A' \to x, Per, \emptyset) : (A \to x, Per, \emptyset) \in P\}$ for permitting grammars.

It is not hard to see that $L(G) = L(G')$.

(2) $\Rightarrow$ (1): Let G be a random context (permitting, forbidding) grammar using definition (2). Construct the grammar G' of the same type using definition (1) so that for each production $p = (A \to x, Per, For)$ of G with $A \notin For$, add $p' = (A \to x, Per - \{A\}, For)$ to productions of G'. Clearly, p is applicable in G if and only if p' is applicable in G'. □

Similarly, we can modify definition (2) of the direct derivation step for left-random context grammars, i.e., for $u, v \in V^*$ and a production $(A \to x, Per, For) \in P$, the relation $uAv \Rightarrow uxv$ holds provided that $Per \subseteq alph(uA)$ and $alph(uA) \cap For = \emptyset$. As above, we refer to those two definitions as to definitions (1) and (2). The reader can see that the implication (2) $\Rightarrow$ (1) of the previous lemma holds for left-random context (left-permitting, left-forbidding) grammars. Thus, definition (2) is weaker than definition (1) in the sense that every left-random context (left-permitting, left-forbidding) grammar using definition (2) can be converted to an equivalent grammar of the same type using definition (1). As left-forbidding grammars generate only context-free languages, we have that these two definitions are equivalent for left-forbidding grammars. In addition, the implication (1) $\Rightarrow$ (2) holds for left-permitting grammars. Thus, we have the following result.

Lemma 2. *Definitions* (1) *and* (2) *are equivalent for left-permitting and left-forbidding grammars.*

However, the construction (1) $\Rightarrow$ (2) does not work for left-random context grammars.

Open problem 2. *Are these two definitions equivalent for left-random context grammars?*

As definition (2) is weaker than definition (1), in the sense mentioned above, we implicitly use definition (1) from now on. However, definition (2) is also discussed.

3.2 (Left-)Random Context Components

Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$, $n \geq 1$, be a CD grammar system with (left-)random context components and consider an f-mode, $f \in \{*, =1, \geq 1\} \cup \{\leq k : k \geq 1\}$. The behavior of Γ is then characterized by choosing a component and applying any of its productions; the cycle is repeated. Thus, Γ behaves as the (left-)random context grammar $G = (N, T, P_1 \cup P_2 \cup \cdots \cup P_n, S)$. The following holds.

Lemma 3. *For $n \geq 1$ and $f \in \{*, =1, \geq 1\} \cup \{\leq k : k \geq 1\}$,*

1. $\boldsymbol{CD_f(RC_\lambda, n) = RC_\lambda}$ *and* $\boldsymbol{CD_f(RC, n) = RC}$,
2. $\boldsymbol{CD_f(FOR_\lambda, n) = FOR_\lambda}$ *and* $\boldsymbol{CD_f(FOR, n) = FOR}$,
3. $\boldsymbol{CD_f(PER_\lambda, n) = PER_\lambda = PER = CD_f(PER, n)}$ [16],
4. $\boldsymbol{CD_f(\ell PER_\lambda, n) = \ell PER_\lambda}$ *and* $\boldsymbol{CD_f(\ell PER, n) = \ell PER}$,
5. $\boldsymbol{CD_f(\ell FOR_\lambda, n) = CF = CD_f(\ell FOR, n)}$ [8].

By the well-known result $\mathbf{RC}_\lambda = \mathbf{RE}$ [6], we have the following result.

Lemma 4. *For $n \geq 1$ and $f \in \{t\} \cup \{=k, \geq k : k \geq 2\}$, $\boldsymbol{CD_f(RC_\lambda, n) = RC_\lambda}$.*

Given a random context grammar without erasing productions, the grammar can be considered as the only component (with productions $(A \to A, \emptyset, \emptyset)$ added, if needed), which gives the following.

Lemma 5. *For $n \geq 1$ and $f \in \{t\} \cup \{=k, \geq k : k \geq 2\}$, $\boldsymbol{RC \subseteq CD_f(RC, n)}$.*

We prove that the other inclusion holds true, too.

Lemma 6. *For $n \geq 1$, $\boldsymbol{CD_t(RC, n) \subseteq RC}$.*

Proof. Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system with n random context components. Construct the random context grammar $G = (\tilde{N} \cup \{\tilde{S}\}, T \cup \{c\}, P', \tilde{S})$, where c, $\tilde{S}$ are new symbols, $c, \tilde{S} \notin T \cup \tilde{N}$, $\tilde{N} = N \cup N' \cup \{[Q_i], \langle p, Q_i \rangle, [p, Q_i], [i] : Q_i \subseteq P_i, p \in Q_i, 1 \leq i \leq n\}$, $N' = \{X' : X \in N\}$, and P' is constructed as follows:

1. For each $(A \to x, Per, For) \in P_i$, add $(A \to x, Per \cup \{[i]\}, For)$ to P'.
2. For $1 \leq i, \ell \leq n$, add to P'

a) $(\tilde{S} \to S[i], \emptyset, \emptyset)$,
b) $([i] \to [P_i], \emptyset, \emptyset)$,
c) $([\emptyset] \to [\ell], \emptyset, \emptyset)$,
d) $([i] \to c, \emptyset, \emptyset)$.

3. For $Q_i \subseteq P_i$ and $p = (A \to x, Per, For) \in Q_i$, $1 \leq i \leq n$, add also to P'

e) $([Q_i] \to [Q_i - \{p\}], \emptyset, \{A\})$,
f) $([Q_i] \to \langle p, Q_i \rangle, \{A\}, \emptyset)$,
g) $(A \to A', \{\langle p, Q_i \rangle\}, \{A'\})$,
h) $(\langle p, Q_i \rangle \to [p, Q_i], \{A'\}, \{X\})$, for $X \in Per$,
i) $(\langle p, Q_i \rangle \to [p, Q_i], \{A', X\}, \emptyset)$, for $X \in For$,
j) $(A' \to A, \emptyset, \emptyset)$,
k) $([p, Q_i] \to [Q_i - \{p\}], \emptyset, \{A'\})$.

The main idea of the proof is to simulate the CD grammar system so that the non-terminal $[i]$ denotes the simulated component, the grammar uses productions of P_i, and in some moment it non-deterministically decides that no other productions of P_i are applicable, see production (2.b). This is then verified by productions in (3) so that a production p is removed from the non-terminal Q_i if it is not applicable. If no productions of P_i are applicable, production (2.c) can change the component.

We prove that $L_t(\Gamma)c = L(G)$. As non-erasing random context grammars are closed under restricted homomorphisms (Lemma 1.3.3 in [6]), we get that there is a random context grammar H such that $L_t(\Gamma) = L(H)$.

The simulation of Γ by G starts with a production constructed in (2.a) applied only once because $\tilde{S}$ does not occur on the right side of any production. Furthermore, every successful derivation is finished by the application of a production constructed in (2.d). If $uAv \Rightarrow_i uxv$ in Γ, for $1 \leq i \leq n$, $A \in N$, $u, v, x \in (N \cup T)^*$, then $uAv[i] \Rightarrow uxv[i]$ in G by $(A \to x, Per \cup \{[i]\}, For)$. If Γ changes components from P_i to P_ℓ, for some $1 \leq i, \ell \leq n$, and the sentential form is $w \in (N \cup T)^*$, then no production from P_i can be applied. This is verified in G by the following sequence of productions: $w[i] \Rightarrow_G w[P_i] \Rightarrow_G^* w[\emptyset] \Rightarrow_G w[\ell]$, by (2.b), (3)*, and (2.c). In more detail, for every $p = (A \to x, Per, For) \in P_i$, if A does not occur in w, then $w[Q_i] \Rightarrow_G w[Q_i - \{p\}]$, by (3.e); otherwise, if $w = uAv$, we need to check that it is not applicable because of permitting and forbidding sets:

$$\begin{aligned} uAv[Q_i] &\Rightarrow_G uAv\langle p, Q_i \rangle && \text{(by (3.f) and (3.g))} \\ &\Rightarrow_G uA'v\langle p, Q_i \rangle \\ &\Rightarrow_G uA'v[p, Q_i] && \text{(by (3.h) or (3.i))} \\ &\Rightarrow_G uAv[p, Q_i] \Rightarrow_G uAv[Q_i - \{p\}] && \text{(by (3.j) and (3.k))} \end{aligned}$$

where (3.h) is applied if there is $X \in Per$ missing in uv, whereas (3.i) is applied if there is $Y \in For$ occurring in uv. As p is not applicable to w in Γ, at least one of (3.h) and (3.i) must be applicable in G.

On the other hand, to prove $L(G) \subseteq L_t(\Gamma)c$, let h be a homomorphism defined as $h(X) = X$, $X \in N \cup T$, $h(A') = A$, $A' \in N'$, $h(B) = \lambda$ otherwise, and let $N_Q = \tilde{N} - (N \cup N')$. We prove that if $yY \Rightarrow zZ$ in G by (2.b), (2.c), (2.d), or (3), then $h(y) \Rightarrow^* h(z)$ in Γ, for $y, z \in (N \cup N' \cup T)^*$, $Y \in N_Q$, and $Z \in N_Q \cup \{c\}$. As yY contains exactly one symbol from N_Q and this is preserved by (2.b), (2.c), and (3), $yY \Rightarrow zZ$ by (2.b), (2.c), (2.d), or (3) implies that $|yY| = |zZ|$. Consider the form of Y: (I) If $Y = [i]$, productions (1), (2.b), and (2.d)

can be applied. (1) is applied if $yY = uAv[i] \Rightarrow uxv[i] = zZ$ in G, which corresponds to $y = uAv \Rightarrow_i uxv = z$ in Γ by $(A \to x, Per, For) \in P_i$. (2.b) implies that $y = z$ and $Z = [P_i]$. (2.d) is the last step of any successful derivation of G replacing the rightmost non-terminal with c; here, $y = z$ and $Z = c$. (II) If $Y = [Q_i]$, $p = (A \to x, Per, For) \in Q_i$ is tested for its non-applicability to y. If (3.e) is applied, A does not occur in y, which implies that p is not applicable; this is remembered by removing p from Q_i. Here, $y = z$ and $Z = [Q_i - \{p\}]$. If (3.f) is applied, then A appears in y, $y = z$, and $Z = \langle p, Q_i \rangle$ (see III below). If (2.c) is applied, then $Q_i = \emptyset$, $y = z$, and $Z = [\ell]$, for some $1 \leq \ell \leq n$. This means that there is no applicable production and the component can be changed to ℓ. (III) $Y = \langle p, Q_i \rangle$, $p = (A \to x, Per, For) \in Q_i$. If (3.g) is applied, then A' does not occur in y, A' occurs in z, $h(y) = h(z)$, and only (3.h), (3.i), and (3.j) can be applied. If (3.h) is applied, A occurs in $h(y)$ and at least one symbol $X \in Per$ is missing in uv, for $h(y) = h(z) = uAv$. Thus, p is not applicable in Γ, and $Z = [p, Q_i]$ (see IV below). Analogously, if (3.i) is applied, then there is $X \in For$ occurring in uv. Again, p is not applicable in Γ, and $Z = [p, Q_i]$. (IV) $Y = [p, Q_i]$, $p = (A \to x, Per, For) \in Q_i$. If (3.j) is applied, then A' occurs in y, $h(y) = h(z) = z$, and $Z = Y$. If (3.k) is applied, then A' does not occur in $y = z$ and $Z = [Q_i - \{p\}]$. Thus, we remember than p is not applicable. This cycle is repeated until the non-terminal $[\emptyset]$ is reached (see II above). As the successful derivation starts by (2.a), the proof proceeds by induction. Thus, for $u[i], v[\ell] \in (N \cup T)^* N_Q$, we have proved that if $u[i] \Rightarrow^* v[\ell]$ in G, then $u \Rightarrow_i^! v$ in Γ by productions from P_i, which completes the proof. □

The following lemma discusses the effect of the remaining two derivation modes.

Lemma 7. *For $n \geq 1$ and $f \in \{=k, \geq k : k \geq 2\}$, $\boldsymbol{CD_f(RC, n) \subseteq RC}$.*

Proof. Let $\Gamma = (N, T, P_1, P_2, \ldots, P_n, S)$ be a CD grammar system with n random context components working in the $\geq k$-mode, for $k \geq 2$. Construct the random context grammar $G = (\tilde{N} \cup \{\tilde{S}\}, T \cup \{c\}, P', \tilde{S})$, where $c, \tilde{S} \notin T \cup \tilde{N}$, $\tilde{N} = N \cup N_{rhs} \cup \{[i,m], \langle i,m \rangle : 1 \leq i \leq n, 0 \leq m \leq k\}$, $N_{rhs} = \{\langle x \rangle : (A \to x, Per, For) \in P_i, 1 \leq i \leq n\}$, and for all P_i, $1 \leq i \leq n$, and all $(A \to x, Per, For) \in P_i$, add the following productions to P':

1. $(\tilde{S} \to S[i,k], \emptyset, \emptyset)$,
2. $(A \to \langle x \rangle, Per \cup \{[i,m]\}, For \cup N_{rhs})$, where $1 \leq m \leq k$
3. $(\langle x \rangle \to x, \{\langle i,m \rangle\}, \emptyset)$, where $0 \leq m \leq k$
4. $([i,k] \to \langle i,k \rangle, \{\langle x \rangle\}, \emptyset)$,
5. $([i,m] \to \langle i,m-1 \rangle, \{\langle x \rangle\}, \emptyset)$, where $1 \leq m \leq k$,
6. $(\langle i,m \rangle \to [i,m], \emptyset, \{\langle x \rangle\})$, where $0 \leq m \leq k$,
7. $([i,0] \to [j,k], \emptyset, \emptyset)$, where $1 \leq j \leq n$,
8. $([i,0] \to c, \emptyset, \emptyset)$.

Each non-terminal of the form $[i,m]$ or $\langle i,m \rangle$ consists of the index, i, of the simulated component of Γ and the counter, m, of the number of productions of P_i which need to be applied by Γ to allow another component to work. They are used to simulate productions of the ith component and to count the number of simulated components, respectively.

To prove that $L_{\geq k}(\Gamma)c \subseteq L(G)$, we demonstrate that if $uAv \Rightarrow_i uxv$ in Γ by a production $(A \to x, Per, For) \in P_i$, then

$$\begin{aligned} uAv[i,m] &\Rightarrow u\langle x\rangle v[i,m] && \text{(by (2))} \\ &\Rightarrow u\langle x\rangle v\langle i,o\rangle && \text{(by (4) or (5))} \\ &\Rightarrow uxv\langle i,o\rangle && \text{(by (3))} \\ &\Rightarrow uxv[i,o] && \text{(by (6))} \end{aligned}$$

where $o = m-1$, for $m < k$, or $o \in \{k, k-1\}$, for $m = k$. If Γ makes $k+\ell$ steps by productions from P_i, for some $\ell \geq 0$, then in $k+\ell$ repetitions of the derivation of G shown above, the first ℓ is made using production (4), while the last k is made using production (5). Furthermore, when Γ changes its component from P_i to P_j, for some $1 \leq i, j \leq n$, then G derives $x[i,0] \Rightarrow x[j,k]$ by production (7). As every derivation of G starts by a production constructed in (1), the proof proceeds by induction. Finally, the last non-terminal is rewritten to c by (8) as the last step of the simulation.

To complete the proof, we demonstrate that $L(G) \subseteq L_{\geq k}(\Gamma)c$. Let $N_Q = \tilde{N} - (N \cup N_{rhs})$, and let h be a homomorphism defined as $h(X) = X$, $X \in N \cup T$, $h(\langle x\rangle) = x$, $\langle x\rangle \in N_{rhs}$, and $h(B) = \lambda$ otherwise. As $\tilde{S} \Rightarrow S[i,k]$ starts the simulation, we prove that if $yY \Rightarrow zZ$ in G, then $h(y) \Rightarrow^* h(z)$ in Γ, for $y, z \in (N \cup N_{rhs} \cup T)^*$, $Y \in N_Q$, and $Z \in N_Q \cup \{c\}$. Clearly, $S[i,k]$ contains one symbol from N_Q and this is preserved by productions (4) and (2) through (7). The last non-terminal, Y, and the occurrence of a symbol from N_{rhs} control the derivation of G in the following way: (I) $Y = [i,m]$, $1 \leq m \leq k$. If y contains no symbol $\langle x\rangle$, (2) simulates the corresponding production of P_i including permitting and forbidding checks, i.e., $y = uAv \Rightarrow u\langle x\rangle v = z$ and $h(y) = uAv \Rightarrow uxv = h(z)$; $Z = [i,m]$. If it contains $\langle x\rangle$, (4) or (5) is applied to count the number of remaining steps of the current component. Thus, $h(y) = h(z)$, and $Z = \langle i,m\rangle$ or $Z = \langle i,m-1\rangle$. (II) $Y = \langle i,m\rangle$, $0 \leq m \leq k$. If $\langle x\rangle$ occurs in y, (3) finishes the simulation of a production from P_i, i.e., $y = u\langle x\rangle v \Rightarrow uxv = z$ and $h(y) = h(z)$; $Z = \langle i,m\rangle$. If there is no $\langle x\rangle$ in y, Y is rewritten by (6), $y = z$, and $Z = [i,m]$. (III) $Y = [i,0]$. Then, either (7) is applied to replace $[i,0]$ with $Z = [j,k]$, for some $1 \leq j \leq n$, or (8) finishes the simulation by replacing $[i,0]$ with $Z = c$. Thus, for $\alpha[i,m] \in (N \cup T)^* N_Q$, we have proved that only the sequence of productions (2), (4) or (5), (3), and (6) is applicable, resulting in a string over $(N \cup T)^* N_Q$. As the derivation starts by (1), generating a string over $(N \cup T)^* N_Q$, the proof proceeds by induction. As, in addition, productions (4) do not change the counter, (5) decrease the counter, and (2) (simulating productions from P_i) are applicable only if the counter is not zero, G simulates at least k applications of productions from P_i. Thus, we have proved that $L(G) = L_{\geq k}(\Gamma)c$ and, again, by [6, Lemma 1.3.3], there exists a random context grammar H such that $L_{\geq k}(\Gamma) = L(H)$.

Finally, by omitting productions constructed in (4), G simulates exactly k applications of productions of the given component. Thus, it proves the statement for $=k$-mode, where $k \geq 2$. Hence, the proof is complete. □

The constructions of proofs of Lemmas 6 and 7 can be modified so that the results also hold for random context components using definition (1). Productions constructed in (3.g) and (3.j) are removed from the construction of the proof of Lemma 6, and symbol A' is

replaced with A in the productions constructed in (3) of that proof. The proof of Lemma 7 holds for definition (1) as it is.

We summarize the results in the following theorem.

Theorem 1. *For $n \geq 1$ and $f \in \{t\} \cup \{=k, \geq k : k \geq 2\}$, $\boldsymbol{CD_f(RC}, n) = \boldsymbol{RC}$.*

3.3 (Left-)Forbidding Components

Considering terminal derivation mode, the following theorem is proved in [9]. Note that definition (2) of the direct derivation step is used there. However, by simple modifications, constructions are also valid for forbidding components using definition (1).

Theorem 2. *For $n \geq 2$, $\boldsymbol{CD_t(FOR_\lambda}, n) = \boldsymbol{RE}$ and $\boldsymbol{CD_t(FOR}, n) = \boldsymbol{RC} \subset \boldsymbol{CS}$.*

The following theorems discuss the remaining modes.

Theorem 3. $\boldsymbol{RC} = \bigcup_{k,n\geq 1} \boldsymbol{CD_{=k}(FOR}, n)$ *and* $\boldsymbol{RC_\lambda} = \bigcup_{k,n\geq 1} \boldsymbol{CD_{=k}(FOR_\lambda}, n)$.

Proof. The inclusions $\mathbf{CD_{=k}(FOR_\lambda}, n) \subseteq \mathbf{RC_\lambda}$ and $\mathbf{CD_{=k}(FOR}, n) \subseteq \mathbf{RC}$ follow by Lemmas 4 and 7, respectively. To prove the other inclusions, let $G = (N, T, P, S)$ be a random context grammar, and construct the CD grammar system Γ with $n = |P| + 1$ forbidding components. The main idea of the proof is to introduce a new component for each production of G in which the presence of all the permitting symbols is verified by replacing them one by one (see more details below). Thus, $\Gamma = (N \cup N' \cup N_{rhs} \cup N_{cnt}, T, P_1, P_2, \ldots, P_n, S)$, where

- $N' = \{A' : A \in N\}$, $N_{rhs} = \{\langle x \rangle : (A \to x, Per, For) \in P\}$,
- $m = \max\{|Per - \{A\}| : (A \to x, Per, For) \in P\}$,
- $N_{cnt} = \bigcup_{i=1}^{m} N^{(i)}$, for $N^{(i)} = \{A^{(i)} : A \in N\}$, $1 \leq i \leq m$,

and Γ works in $=k$-mode, where $k = m + 1$. Let β be a bijection from P to $\{1, 2, \ldots, |P|\}$. The components of Γ are constructed as follows:

1. For each production $(A \to x, Per, For) \in P$, add to $P_{\beta(A\to x, Per, For)}$:
   a) $(A \to A^{(m-|Per-\{A\}|)}, \emptyset, N_{cnt} \cup N_{rhs} \cup N')$, for $m - |Per - \{A\}| > 0$,
   b) $(A^{(i)} \to A^{(i-1)}, \emptyset, N_{rhs} \cup N')$, where $1 < i \leq m - |Per - \{A\}|$,
   c) $(B \to B', \emptyset, N_{rhs} \cup \{B'\})$, where $B \in Per - \{A\}$,
   d) $(A^{(1)} \to \langle x \rangle, \emptyset, For \cup N_{rhs})$,
   e) $(A \to \langle x \rangle, \emptyset, For \cup N_{rhs})$, for $m - |Per - \{A\}| = 0$.
2. To P_n add:
   a) $(B' \to B, \emptyset, N_{cnt})$, where $B \in N$,
   b) $(\langle x \rangle \to \langle x \rangle, \emptyset, N' \cup N_{cnt})$, where $\langle x \rangle \in N_{rhs}$.
   c) $(\langle x \rangle \to x, \emptyset, N' \cup N_{cnt})$, where $\langle x \rangle \in N_{rhs}$,

Note that the crucial point of the construction is that exactly $|Per - \{A\}|$ productions of type (1.c) have to be applied, where the forbidding context guarantees that only one occurrence of any nonterminal can be primed.

To prove $L(G) = L_{=k}(\Gamma)$, we show that each component P_i, $i = 1,2,\ldots,n-1$, simulates exactly one production of G. Let $p = (A \to x, Per, For)$ be a production of G. The simulation is done by a sequence of productions from $P_{\beta(p)}$ as follows. (A) For $m - |Per - \{A\}| > 0$: A production constructed in (1.a), then $m - |Per - \{A\}| - 1$ productions constructed in (1.b), then $|Per - \{A\}|$ productions constructed in (1.c), finished by a production constructed in (1.d). Summarized, $1 + (m - |Per - \{A\}| - 1) + (|Per - \{A\}|) + 1 = m + 1$ productions are applied, and the component is changed. (B) For $m - |Per - \{A\}| = 0$: $|Per - \{A\}|$ productions constructed in (1.c), finished by a production constructed in (1.e). Again, $|Per - \{A\}| + 1 = m - |Per - \{A\}| + |Per - \{A\}| + 1 = m + 1$ productions are applied, and the component is changed. After that, the current sentential form contains $|Per - \{A\}|$ primed non-terminals and one symbol $\langle x \rangle$, for some x. The following sequence of productions of P_n is applied to remove these symbols: $|Per - \{A\}|$ productions constructed in (2.a), $m - |Per - \{A\}|$ productions constructed in (2.b), and a production constructed in (2.c).

On the other hand, we show that this is the only possible behavior of Γ. (A) For $m - |Per - \{A\}| > 0$: Considering the work of P_i, a production constructed in (1.a) has to be applied first; otherwise, if (1.c) is applied first, then (1.a) (and also (1.b) and (1.d)) cannot be applied. However, as $|Per - \{A\}| < m$ and P_i has to perform $m + 1$ steps, the derivation is blocked. (B) For $m - |Per - \{A\}| = 0$: Considering the work of P_i, if $|Per - \{A\}| > 0$, then productions constructed in (1.c) have to be applied first; otherwise, if (1.e) is applied first, then (1.c) cannot be applied. However, as $|Per - \{A\}| = m > 0$, the derivation is blocked. Thus, each component P_i, $i = 1,2,\ldots,n-1$, generates no more than m primed non-terminals and one symbol $\langle x \rangle$, for some x. Considering the component P_n, productions constructed in (2.a) have to be applied first. Then, if (2.c) is not applied as the $(m+1)$st production, the derivation is blocked. On the other hand, if (2.c) is not applied at all, i.e., only (2.b) is applied, then only P_n contains applicable productions. Thus, until a production constructed in (2.c) is applied as the $(m+1)$st production, P_n is chosen repeatedly to work.

As we do not introduce any new erasing productions, the proof is complete. □

Corollary 1. *$RC = \bigcup_{k,n\geq 1} CD_{\geq k}(FOR, n)$ and $RC_\lambda = \bigcup_{k,n\geq 1} CD_{\geq k}(FOR_\lambda, n)$.*

Proof. In the proof of the previous theorem, each component P_i, $i = 1,2,\ldots,n-1$, performs exactly k steps, only P_n can perform more than k steps. □

Corollary 2.

1. *$RC = \bigcup_{n\geq 1} CD_{=2}(FOR, n) = \bigcup_{n\geq 1} CD_{\geq 2}(FOR, n)$, and*
2. *$RC_\lambda = \bigcup_{n\geq 1} CD_{=2}(FOR_\lambda, n) = \bigcup_{n\geq 1} CD_{\geq 2}(FOR_\lambda, n)$.*

Proof. It is shown in [5] that, for any random context grammar G', there is an equivalent random context grammar $G = (N, T, P, S)$ using definition (2) such that G is with erasing productions if and only if G' is, and for $(A \to w, Per, For) \in P$, $|Per \cup For| = 1$. Thus, in the proof of the previous theorem, we have $m \leq 1$, and if $m = 1$, then Γ works in =2-mode. □

Open problem 3. *Can the number of components be reduced?*

The following is proved for left-forbidding CD grammar systems in [8].

Theorem 4. *For $n \geq 2$ and $f \in \{t\} \cup \{=k, \geq k : k \geq 2\}$, $\boldsymbol{RE} = \boldsymbol{CD}_f(\ell\boldsymbol{FOR}_\lambda, n)$ and $\boldsymbol{CS} = \boldsymbol{CD}_f(\ell\boldsymbol{FOR}, n)$. In addition, any recursively enumerable language can be generated by a left-forbidding CD grammar system working in terminal derivation mode with two components and twelve non-terminals.*

3.4 (Left-)Permitting Components

Although the relation between the families **PER** and **FOR** is not known, permitting CD grammar systems and forbidding CD grammar systems working in terminal derivation mode are of the same power [4].

Theorem 5. *For $n \geq 2$, $\boldsymbol{CD}_t(\boldsymbol{PER}_\lambda, n) = \boldsymbol{CD}_t(\ell\boldsymbol{PER}_\lambda, n) = \boldsymbol{RE}$ and $\boldsymbol{CD}_t(\boldsymbol{PER}, n) = \boldsymbol{RC} \subset \boldsymbol{CS} = \boldsymbol{CD}_t(\ell\boldsymbol{PER}, n)$. In addition, any recursively enumerable language can be generated by a left-permitting CD grammar system working in terminal derivation mode with six components and nineteen non-terminals.*

Open problem 4. *What is the generative power of permitting (left-permitting) CD grammar systems working in the f-mode, for $f \in \{=k, \geq k : k \geq 2\}$?*

In this paper, the components use definition (1) of the direct derivation step. In comparison with forbidding CD grammar systems working in terminal derivation mode where these two definitions are equivalent, we do not know whether they are equivalent in the case of permitting CD grammar systems with at least two components, although they are equivalent for permitting grammars (see Lemma 1).

Open problem 5.

1. *What is the power of permitting CD grammar systems if the components (permitting grammars) use definition (2) of the direct derivation step?*
2. *What is the power of left-permitting CD grammar systems if the components use definition (2)?*

4 Conclusion and discussion

In this paper, we have discussed CD grammar systems where components are (variants of) random context grammars. Recall that CD grammar systems with only permitting and with only forbidding components have been studied in [4] and [9], respectively. Originally, the components of forbidding CD grammar systems used definition (2) of the direct derivation step. However, the constructions can be modified so that the results hold for forbidding CD grammar systems with components using definition (1) as well. In addition, all the results concerning CD grammar systems with random context components proved in this paper hold for both definitions. On the other hand, to achieve results concerning the generative

power of CD grammar systems with permitting components proved in [4], definition (1) of the direct derivation step is used. Unfortunately, in this case, we do not know whether the same results can also be achieved for CD grammar systems with permitting components using definition (2), although definitions (1) and (2) are equivalent for permitting grammars (see Lemma 1). Note that it is not hard to see that definition (1) allows us to check for at least two occurrences of a given non-terminal symbol (the rewritten one and the one occurring on the left or on the right of the rewritten symbol), while definition (2) seems to be too weak to check for that property. However, it might be possible to check for that property by using some mechanisms of CD grammar systems instead. The cases of CD grammar systems with left-permitting and left-forbidding grammars as components are studied in [4] and [8], respectively. In these cases, only definition (1) of the direct derivation step is considered. An overview of the results follows.

For any $n \geq 1$, derivation modes $f \in \{*, =1, \geq 1\} \cup \{\leq k : k \geq 1\}$, and language families $\mathbf{X} \in \{\mathbf{RC}_\lambda, \mathbf{RC}, \mathbf{FOR}_\lambda, \mathbf{FOR}, \mathbf{PER}_\lambda, \mathbf{PER}, \ell\mathbf{RC}_\lambda, \ell\mathbf{RC}, \ell\mathbf{PER}_\lambda, \ell\mathbf{PER}, \ell\mathbf{FOR}_\lambda, \ell\mathbf{FOR}\}$,

$$\mathbf{CD}_f(\mathbf{X}, n) = \mathbf{X},$$

and for any $n \geq 2$ and derivation modes $f \in \{t\} \cup \{=k, \geq k : k \geq 2\}$,

1. $\mathbf{CD}_f(\mathbf{RC}_\lambda, n) = \bigcup_{m \geq 1} \mathbf{CD}_f(\mathbf{FOR}_\lambda, m) = \mathbf{CD}_f(\ell\mathbf{RC}_\lambda, n) = \mathbf{CD}_f(\ell\mathbf{FOR}_\lambda, n) = \mathbf{RE}$,
2. $\mathbf{CD}_t(\mathbf{FOR}_\lambda, n) = \mathbf{CD}_t(\mathbf{PER}_\lambda, n) = \mathbf{CD}_t(\ell\mathbf{PER}_\lambda, n) = \mathbf{RE}$,
3. $\mathbf{CD}_f(\ell\mathbf{RC}, n) = \mathbf{CD}_f(\ell\mathbf{FOR}, n) = \mathbf{CD}_t(\ell\mathbf{PER}, n) = \mathbf{CS}$,
4. $\mathbf{CD}_f(\mathbf{RC}, n) = \bigcup_{m \geq 1} \mathbf{CD}_f(\mathbf{FOR}, m) = \mathbf{RC}$,
5. $\mathbf{CD}_t(\mathbf{FOR}, n) = \mathbf{CD}_t(\mathbf{PER}, n) = \mathbf{RC}$.

Recall that it has recently been shown in [16] that the generative power of permitting grammars coincides with the generative power of non-erasing permitting grammars. In other words, the erasing productions can be removed from permitting grammars. In addition, although left-permitting grammars are similar to permitting grammars, it is an open problem whether a similar relation holds for left-permitting grammars with and without erasing productions. Note that the proof from [16] cannot be directly applied to the case of left-permitting grammars because it uses the following property of permitting grammars that does not hold for left-permitting grammars: for any permitting production $(A \to w, U)$, if the production can be used to one of the occurrences of A, then it can be used to any occurrence of A in the sentential form.

Finally, given two families of grammars generating the same family of languages, an interesting question is whether or when it is also the case that the language families generated by CD grammar systems, using these two types of grammars as components, also coincide. From the results mentioned above, we can immediately see that, e.g., although $\mathbf{CF} = \ell\mathbf{FOR} = \ell\mathbf{FOR}_\lambda$, we have that for any $n \geq 2$, $\mathbf{CD}_t(\mathbf{CF}, n) \subset \mathbf{CD}_t(\ell\mathbf{FOR}, n) \subset \mathbf{CD}_t(\ell\mathbf{FOR}_\lambda, n)$. Similarly, although $\mathbf{PER} = \mathbf{PER}_\lambda$, the proper inclusion $\mathbf{CD}_t(\mathbf{PER}, n) \subset \mathbf{CD}_t(\mathbf{PER}_\lambda, n)$ holds for any $n \geq 2$. On the other hand, it is obvious that the equality of language families generated by CD grammar systems with different types of components does not imply that the language families generated by grammars of these types coincide. Thus, the question when the same power of components implies the same power of CD

grammar systems is open. Moreover, note that the discussion in Section 3.1 can also be considered in this way.

Acknowledgments

The authors gratefully acknowledge very useful suggestions and comments of the anonymous referees.

The research by the first author has been supported by the MŠMT Research Plan no. MSM0021630528, and by the MŠMT grants 2C06008 and MEB041003. The research by the second author has partially been supported by the Czech Academy of Sciences, Institutional Research Plan no. AV0Z10190503, and by the GAČR grant no. 202/11/P028.

References

[1] Bordihn, H. and Fernau, H. Accepting grammars and systems: An overview. In *Developments in Language Theory*, pages 199–208, 1995. Technical Report 9/94, Universität Karlsruhe, Fakultät für Informatik, 1994.

[2] Csuhaj-Varjú, E. and Dassow, J. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics (EIK)*, 26(1-2):49–63, 1990.

[3] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., and Păun, Gh. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Topics in Computer Mathematics 5, Yverdon, 1994.

[4] Csuhaj-Varjú, E., Masopust, T., and Vaszil, Gy. Cooperating distributed grammar systems with permitting grammars as components. *Romanian Journal of Information Science and Technology*, 12(2):175–189, 2009.

[5] Dassow, J. and Masopust, T. On restricted context-free grammars. In *Developments in Language Theory*, pages 434–435, 2010.

[6] Dassow, J. and Păun, Gh. *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.

[7] Ewert, S. and van der Walt, A. P. J. A pumping lemma for random permitting context languages. *Theoretical Computer Science*, 270(1–2):959–967, 2002.

[8] Goldefus, F., Masopust, T., and Meduna, A. Left-forbidding cooperating distributed grammar systems. *Theoretical Computer Science*, 411(40-42):3661–3667, 2010.

[9] Masopust, T. On the terminating derivation mode in cooperating distributed grammar systems with forbidding components. *Internation Journal of Foundations of Computer Science*, 20(2):331–340, 2009.

[10] Mihalache, V. Programmed grammar systems. In *Developments in Language Theory*, pages 430–437, 1993.

[11] Păun, Gh. A variant of random context grammars: Semi-conditional grammars. *Theoretical Computer Science*, 41:1–17, 1985.

[12] Rozenberg, G. and Salomaa, A., editors. *Handbook of Formal Languages*, volume 1–3. Springer, Berlin, 1997.

[13] Salomaa, A. *Formal languages*. Academic Press, New York, 1973.

[14] van der Walt, A. P. J. Random context grammars. In *Proceedings of the Symposium on Formal Languages*, pages 163–165, 1970.

[15] van der Walt, A. P. J. and Ewert, S. A shrinking lemma for random forbidding context languages. *Theoretical Computer Science*, 237(1-2):149–158, 2000.

[16] Zetzsche, G. On erasing productions in random context grammars. In *International Colloquium on Automata, Languages and Programming (2)*, pages 175–186, 2010.

Received 24th of June 2010

Acta Cybernetica **20** (2011) 285–329.

Weak Functional Dependencies on Trees with Restructuring

Attila Sali* and Klaus-Dieter Schewe†

Abstract

We present an axiomatisation for weak functional dependencies, i.e. disjunctions of functional dependencies, in the presence of several constructors for complex values. The investigated constructors capture records, sets, multisets, lists, disjoint union and optionality, i.e. the complex values are indeed trees. The constructors cover the gist of all complex value data models including object oriented databases and XML. Functional and weak functional dependencies are expressed on a lattice of subattributes, which even carries the structure of a Brouwer algebra as long as the union-constructor is absent. Its presence, however, complicates all results and proofs significantly. The reason for this is that the union-constructor causes non-trivial restructuring rules to hold. In particular, if either the set- or the the union-constructor is absent, a subset of the rules is complete for the implication of ordinary functional dependencies, while in the general case no finite axiomatisation for functional dependencies exists.

Keywords: functional dependency, weak functional dependency, axiomatisation, complex values, restructuring, embedded dependency, rational tree

1 Introduction

In the relational data model (RDM) a lot of research has been spent on the theory of dependencies, i.e. first-order sentences that are supposed to hold for all database instances (see [3, 25]). Various classes of dependencies for the RDM have been introduced (see [32] for a survey), and large parts of database theory deals with the finite axiomatisation of these dependencies and the finite implication problem for them. That is to decide that a dependency φ is implied by a set of dependencies Σ, where implication refers to the fact that all finite models of Σ are also models of φ. The easiest, yet most important class of dependencies is the class of *functional*

*Alfréd Rényi Institute of Mathematics, Budapest, P.O.B.127, H-1364 Hungary, E-mail: sali@renyi.hu

†Software Competence Center Hagenberg, Hagenberg, Austria and Johannes-Kepler-University Linz, Research Institute for Applied Knowledge Processing, Linz, Austria, E-mail: kd.schewe@scch.at, kd.schewe@faw.at

dependencies (FDs). Armstrong (see [6]) was the first to give a finite axiomatisation for FDs.

Dependency theory is a cornerstone of database design, as the semantics of the application domain cannot be expressed only by structures. Database theory has to investigate the implications arising from the presence of dependencies. This means to describe semantically desirable properties of "well-designed" databases, e.g. the absence of redundancy, to characterise them (if possible) syntactically by in-depth investigation of the dependencies, and to develop algorithms to transform schemata into normal forms, which guarantee the desirable properties to be satisfied.

However, the field of databases is no longer the unique realm of the RDM. First, so called semantic data models have been developed (see e.g. [9, 22]), which were originally just meant to be used as design aids, as application semantics was assumed to be easier captured by these models (see the argumentation in [7, 10, 35]). Later on some of these models, especially the nested relational model (see e.g. [25]), object oriented models (see e.g. [30]) and object-relational models, the gist of which are captured by the higher-order Entity-Relationship model (HERM, see [33, 34]) have become interesting as data models in their own right and some dependency and normalisation theory has been carried over to these advanced data models (see [14, 23, 24, 25, 31] as samples of the many work done on this so far). Most recently, the major research interest is on the model of semi-structured data and XML (see e.g. [1]), which may also be regarded as some kind of object oriented model.

We refer to all these models as *higher-order data models*. This is, because the most important extension that came with these models was the introduction of constructors for complex values. These constructors usually comprise bulk constructors for sets, lists and multisets, a disjoint union constructor, and an optionality or null-constructor. In fact, all the structure of higher-order data models (including XML as far as XML can be considered a data model) is captured by the introduction of (some or all of) these constructors.

The key problem is to develop dependency theories (or preferably a unified theory) for the higher-order data models. The development of such a dependency theory will have a significant impact on understanding application semantics and laying the grounds for a logically founded theory of well-designed non-relational databases.

So far, mainly keys and FDs for advanced data models have been investigated (see [5, 8, 12, 13, 15, 19, 20, 26, 27, 37, 38]), and this has led to several normal form proposals (see [4, 5, 16, 37]). The work in [16] contains explicit definitions of redundancy and update anomalies and proves (in the spirit of the work in [36]) that the suggested higher-level normal form (HLNF) in the presence of FDs is indeed equivalent to the absence of redundancy and sufficient for the absence of update anomalies. The work in [18] deals with disjunctions of FDs leading to so-called weak functional dependencies (wFDs), while in [17], [21], [39] and [40] first attempts are made to generalise multi-valued dependencies.

The work in this article still deals with functional dependencies and weak functional dependencies, in particular with the axiomatisation problem. The motivation for this work is that all the approaches made so far only deal with part of the

problem. In other words, we still do not have one coherent theory, but merely a patchwork of partial (though nevertheless non-trivial) results:

- The different approaches use different definitions of functional dependencies none of which subsumes the other ones. Arenas and Libkin (see [5]) and similarly Vincent and Liu (see [37]) formalise FDs using paths in XML trees, while Hartmann et al. (see [19]) exploit constructors for lists, disjoint unions and optionality. Despite some initial attempts (see e.g. [41]) so far no common framework subsuming all these different classes of FDs exists. In particular, the class of FDs in [19] has a finite axiomatisation, while the one investigated in [5] has not.

- No approach so far deals with all mentioned constructors at the same time. Hartmann et al. (see [20]) prove a finite axiomatisation taking all constructors into account except the disjoint union constructor. The proof exploits the underlying algebraic structure of Brouwer algebras. Hartmann et al. (see [19]) prove a finite axiomatisation taking all but the set and multiset constructors into account, but at the same time deal with embedded functional dependencies and recursion. Finally, Sali and Schewe (see [27]) take all constructors into account and prove a finite axiomatisation for a restricted class of FDs, which still subsumes the one in [20].

The first objective of the research reported in this article was to remove the remaining restrictions in previous work (see [27]) and to achieve a finite axiomatisation for FDs on models, in which all constructors are present. We will show that such an axiomatisation does not exist. More precisely, we show that we have non-axiomatisability, if the set and the union constructor are combined, whereas if one of them is absent, we obtain a finite axiomatisation. However, switching to the slightly extended class of weak functional dependencies we obtain a finite, though not k-ary axiomatisation. This axiomatisation contains a large number of structural axioms reflecting the non-trivial equivalences between subattributes, which caused significant challenges for the completeness proof. These equivalences result from restructuring rules, which were mostly known already long ago (see e.g. [2]).

Our second objective was to provide a framework that subsumes the existing approaches to dependency theory at outlined below. For this we extend the framework of nested attributes resulting from the various constructors, which in fact captures finite trees, to rational trees, i.e. we capture recursion. Furthermore, we deal with wFDs and FDs that are defined on embedded attributes. With these extensions the classes of FDs developed by Arenas, Libkin and Vincent, Liu, respectively, can be represented as special cases of the general class of FDs. The axiomatisation of the enlarged class of wFDs is straightforward, once the axiomatisation of wFDs in the presence of all constructors is known.

Overview

In Section 2 we define the preliminaries for our theory of wFDs. We start with the definition of nested attributes that are composed of simple attributes using the constructors that have been mentioned above. Each nested attribute defines a set of complex values called its domain, and each complex value can be represented as a finite tree. We then define subattributes, which give rise to canonical projection maps on the domains. The presence of the union constructor leads to restructuring rules, which define non-trivial equivalences the set of subattributes of a given nested attribute. Finally, we investigate the algebraic structure of the set of subattributes of a given nested attribute. We obtain a lattice, which is even a Brouwer algebra, if the union constructor is absent. Nevertheless, also in the general case it is advantageous to define the notion of relative pseudo-complement.

In Section 3 we study certain ideals in such lattices of subattributes, focusing on the set of subattributes, on which two complex values coincide. These ideals are therefore called *coincidence ideals*. The objective is to obtain a precise characterisation in the sense that whenever an ideal satisfies the given set of properties, we can guarantee the existence of two complex values that coincide exactly on the given ideal. This leads to the *Central Theorem* on coincidence ideals, which will be a cornerstone of the completeness proof. The proof of this result, however, appears in [28].

In Section 4 we introduce FDs and wFDs formally and first derive sound derivation rules, most of which are structural axioms reflecting the properties of coincidence ideals. The main result in this section will be the *Completeness Theorem* for the implication of wFDs. We then approach the simpler class of FDs and first show the completeness of a subset of the rules in case not both the set and the union constructors are used. If both appear together, we show non-axiomatisability. Thus, the results in Section 4 fulfil our first objective.

In Section 5 we approach our second objective. We first introduce embedded dependencies and show that they do not affect our axiomatisation of wFDs. In a second step we extend the definition of nested attributes capturing also rational tree values, as they are used in the object models (see e.g. [3] and [30]). We will show that the axiomatisation of wFDs will also be preserved by this extension.

In Section 6 we discuss the relationship with related work. We show that the classes of FDs defined by Arenas, Libkin and Vincent, Liu, respectively, are captured in our framework with all extensions discussed. We discuss the impact of this result.

Finally, we summarise our work and discuss conclusions in Section 7. This includes a brief discussion of additional restructuring rules, problems of keys and Armstrong instances, and an outlook on other classes of dependencies.

2 Algebras of Nested Attributes

In this section we define our model of nested attributes, which covers the gist of higher-order data models including XML. In particular, we investigate the structure

of the set $\mathcal{S}(X)$ of subattributes of a given nested attribute X. We show that we obtain a lattice, which in general is non-distributive. This lattice becomes a Brouwer algebra, if the union constructor is not used.

2.1 Nested Attributes

We start with a definition of simple attributes and values for them.

Definition 1. A *universe* is a finite set $\mathcal{U}$ together with domains (i.e. sets of values) $dom(A)$ for all $A \in \mathcal{U}$. The elements of $\mathcal{U}$ are called *simple attributes.*

For the relational model a universe was sufficient, as a relation schema could be defined by a subset $R \subseteq \mathcal{U}$. For higher-order data models, however, we need nested attributes. In the following definition we use a set $\mathcal{L}$ of labels, and tacitly assume that the symbol λ is neither a simple attribute nor a label, i.e. $\lambda \notin \mathcal{U} \cup \mathcal{L}$, and that simple attributes and labels are pairwise different, i.e. $\mathcal{U} \cap \mathcal{L} = \emptyset$.

Definition 2. Let $\mathcal{U}$ be a universe and $\mathcal{L}$ a set of labels. The set $\mathcal{N}$ of *nested attributes* (over $\mathcal{U}$ and $\mathcal{L}$) is the smallest set with $\lambda \in \mathcal{N}$, $\mathcal{U} \subseteq \mathcal{N}$, and satisfying the following properties:

- for $X \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X(X'_1, \ldots, X'_n) \in \mathcal{N}$;
- for $X \in \mathcal{L}$ and $X' \in \mathcal{N}$ we have $X\{X'\} \in \mathcal{N}$, $X[X'] \in \mathcal{N}$, and $X\langle X'\rangle \in \mathcal{N}$;
- for $X_1, \ldots, X_n \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \in \mathcal{N}$.

We call λ a *null attribute*, $X(X'_1, \ldots, X'_n)$ a *record attribute*, $X\{X'\}$ a *set attribute*, $X[X']$ a *list attribute*, $X\langle X'\rangle$ a *multiset attribute* and $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ a *union attribute.*

In the following we will overload the use of symbols such as X, Y, etc. for nested attributes and labels. As record, set, list and multiset attributes have a unique leading label, this will not cause problems anyway. In all other cases it is clear from the context, whether a symbol denotes a nested attribute in $\mathcal{N}$ or a label. Usually, labels never appear as stand-alone symbols.

We also take the freedom to change the leading label X in a set, list or multiset attribute to $X_{\{1,\ldots,n\}}$, if the component attribute is a union attribute, say $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$. This emphasises the factors in the union attribute. We will see in the next two subsections that this notation will become important, when restructuring is considered.

We can now extend the association *dom* from simple to nested attributes, i.e. for each $X \in \mathcal{N}$ we will define a set of values $dom(X)$.

Definition 3. For each nested attribute $X \in \mathcal{N}$ we get a *domain* $dom(X)$ as follows:

- $dom(\lambda) = \{\top\}$;

- $dom(X(X'_1,\ldots,X'_n)) = \{(v_1,\ldots,v_n) \mid v_i \in dom(X'_i) \text{ for } i = 1,\ldots,n\}$;
- $dom(X\{X'\}) = \{\{v_1,\ldots,v_k\} \mid k \in \mathbb{N} \text{ and } v_i \in dom(X') \text{ for } i = 1,\ldots,k \text{ and } v_i \neq v_j \text{ for } i \neq j\}$, i.e. each element in $dom(X\{X'\})$ is a finite set with (pairwise different) elements in $dom(X')$;
- $dom(X[X']) = \{[v_1,\ldots,v_k] \mid k \in \mathbb{N} \text{ and } v_i \in dom(X') \text{ for } i = 1,\ldots,k\}$, i.e. each element in $dom(X[X'])$ is a finite (ordered) list with (not necessarily different) elements in $dom(X')$;
- $dom(X\langle X'\rangle) = \{\langle v_1,\ldots,v_k\rangle \mid k \in \mathbb{N} \text{ and } v_i \in dom(X') \text{ for } i = 1,\ldots,k\}$, i.e. each element in $dom(X\langle X'\rangle)$ is a finite multiset with elements in $dom(X')$, or in other words each $v \in dom(X')$ has a *multiplicity* $m(v) \in \mathbb{N}$ in a value in $dom(X\langle X'\rangle)$;
- $dom(X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)) = \{(X_i : v_i) \mid v_i \in dom(X'_i) \text{ for } i = 1,\ldots,n\}$.

Note that the relational model is covered, if only the record constructor is used. Thus, instead of a relation schema R we will now consider a nested attribute X, assuming that the universe $\mathcal{U}$ and the set of labels $\mathcal{L}$ are fixed. Instead of an R-relation r we will consider a finite set $r \subseteq dom(X)$.

Further note that each complex value $v \in dom(X)$ for some nested attribute $X \in \mathcal{N}$ can be represented as a finite tree. This will be extended in Section 5 to rational trees.

2.2 Subattributes

In the relational model a functional dependency $X \to Y$ for $X, Y \subseteq R \subseteq \mathcal{U}$ is satisfied by an R-relation r iff any two tuples $t_1, t_2 \in r$ that coincide on all the attributes in X also coincide on the attributes in Y. Crucial to this definition is that we can project R-tuples to subsets of attributes.

Therefore, in order to define FDs on a nested attribute $X \in \mathcal{N}$ we need a notion of subattribute. For this we define a partial order $\geq$ on nested attributes in such a way that whenever $X \geq Y$ holds, we obtain a canonical projection $\pi^X_Y : dom(X) \to dom(Y)$. However, this partial order has to be defined on equivalence classes of attributes, as some domains may be identified.

Definition 4. $\equiv$ is the smallest *equivalence relation* on $\mathcal{N}$ satisfying the following properties:

- $\lambda \equiv X()$;
- $X(X'_1,\ldots,X'_n) \equiv X(X'_1,\ldots,X'_n,\lambda)$;
- $X(X'_1,\ldots,X'_n) \equiv X(X'_{\sigma(1)},\ldots,X'_{\sigma(n)})$ for any permutation $\sigma \in \mathbf{S}_n$;
- $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \equiv X_{\sigma(1)}(X'_{\sigma(1)}) \oplus \cdots \oplus X_{\sigma(n)}(X'_{\sigma(n)})$ for any permutation $\sigma \in \mathbf{S}_n$;

- $X(X_1', \ldots, X_n') \equiv X(Y_1, \ldots, Y_n)$ if $X_i' \equiv Y_i$ for all $i = 1, \ldots, n$;
- $X_1(X_1') \oplus \cdots \oplus X_n(X_n') \equiv X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ if $X_i' \equiv Y_i$ for all $i = 1, \ldots, n$;
- $X\{X'\} \equiv X\{Y\}$ if $X' \equiv Y$;
- $X[X'] \equiv X[Y]$ if $X' \equiv Y$;
- $X\langle X' \rangle \equiv X\langle Y \rangle$ if $X' \equiv Y$;
- $X(X_1', \ldots, Y_1(Y_1') \oplus \cdots \oplus Y_m(Y_m'), \ldots, X_n') \equiv Y_1(X_1', \ldots, Y_1', \ldots, X_n') \oplus \ldots \cdots \oplus Y_m(X_1', \ldots, Y_m', \ldots, X_n')$;
- $X_{\{1,\ldots,n\}}\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\} \equiv X_{\{1,\ldots,n\}}(X_1\{X_1'\}, \ldots, X_n\{X_n'\})$;
- $X_{\{1,\ldots,n\}}\langle X_1(X_1') \oplus \cdots \oplus X_n(X_n') \rangle \equiv X_{\{1,\ldots,n\}}(X_1\langle X_1' \rangle, \ldots, X_n\langle X_n' \rangle)$.

Basically, the first four cases in this equivalence definition state that λ in record attributes can be added or removed, and that order in record and union attributes does not matter. The last three cases in Definition 4 cover restructuring rules, two of which were already introduced by Abiteboul and Hull (see [2]). Obviously, if we have a set of labelled elements with up to n different labels, we can split this set into n subsets, each of which contains just the elements with a particular label, and the union of these sets is the original set. The same holds for multisets. Of course, we can also split a list of labelled elements into lists containing only elements with the same label, thereby preserving the order, but in this case we cannot invert the splitting and thus cannot claim an equivalence.

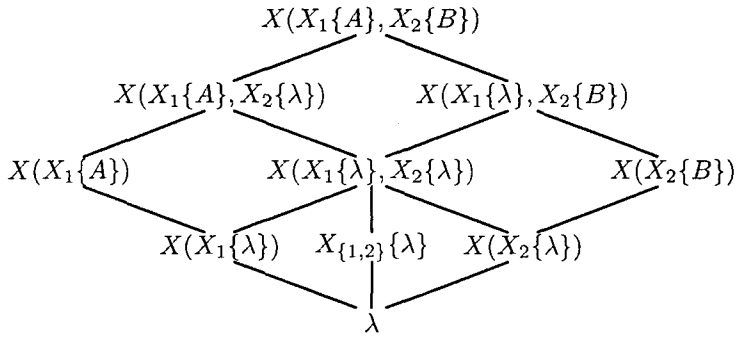


Figure 1: The lattice $\mathcal{S}(X\{X_1(A) \oplus X_2(B)\}) = \mathcal{S}(X(X_1\{A\}, X_2\{B\}))$

In the following we identify $\mathcal{N}$ with the set $\mathcal{N}/_{\equiv}$ of equivalence classes. In particular, we will write $=$ instead of $\equiv$, and in the following definition we should say that Y is a subattribute of X iff $\tilde{X} \geq \tilde{Y}$ holds for some $\tilde{X} \equiv X$ and $\tilde{Y} \equiv Y$. In particular, for $X \equiv Y$ we obtain $X \geq Y$ and $Y \geq X$.

Definition 5. For $X, Y \in \mathcal{N}$ we say that Y is a *subattribute* of X, iff $X \geq Y$ holds, where $\geq$ is the smallest partial order on $\mathcal{N}/_{\equiv}$ satisfying the following properties:

- $X \geq \lambda$ for all $X \in \mathcal{N}$;
- $X(Y_1, \ldots, Y_n) \geq X(X'_{\sigma(1)}, \ldots, X'_{\sigma(m)})$ for some injective $\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $Y_{\sigma(i)} \geq X'_{\sigma(i)}$ for all $i = 1, \ldots, m$;
- $X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \geq X_{\sigma(1)}(X'_{\sigma(1)}) \oplus \cdots \oplus X_{\sigma(n)}(X'_{\sigma(n)})$ for some permutation $\sigma \in \mathbf{S}_n$ and $Y_i \geq X'_i$ for all $i = 1, \ldots, n$;
- $X\{Y\} \geq X\{X'\}$ iff $Y \geq X'$;
- $X[Y] \geq X[X']$ iff $Y \geq X'$;
- $X\langle Y \rangle \geq X\langle X' \rangle$ iff $Y \geq X'$;
- $X_{\{1,\ldots,n\}}[X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)] \geq X(X_1[X'_1], \ldots, X_n[X'_n])$;
- $X_{\{1,\ldots,k\}}[X_1(X'_1) \oplus \cdots \oplus X_k(X'_k)] \geq X_{\{1,\ldots,\ell\}}[X_1(X'_1) \oplus \cdots \oplus X_\ell(X'_\ell)]$ for $k \geq \ell$;
- $X(X_{i_1}\{\lambda\}, \ldots, X_{i_k}\{\lambda\}) \geq X_{\{i_1,\ldots,i_k\}}\{\lambda\}$;
- $X(X_{i_1}\langle\lambda\rangle, \ldots, X_{i_k}\langle\lambda\rangle) \geq X_{\{i_1,\ldots,i_k\}}\langle\lambda\rangle$;
- $X(X_{i_1}[\lambda], \ldots, X_{i_k}[\lambda]) \geq X_{\{i_1,\ldots,i_k\}}[\lambda]$.

Note that the last four cases in Definition 5 cover further restructuring rules due to the union constructor. Obviously, if we are given a list of elements labelled with $X_1, \ldots, X_n$, we can take the individual sublists – preserving the order – that contain only those elements labelled by X_i and build the tuple of these lists. In this case we can turn the label into a label for the whole sublist. This explains the first of the last four subattribute relationships.

For the other restructuring rules we have to add a little remark on notation here. As we identify $X\{X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)\}$ with $X(X_1\{X'_1\}, \ldots, X_n\{X'_n\})$, we obtain subattributes $X(X_{i_1}\{X'_{i_1}\}, \ldots \ldots, X_{i_k}\{X'_{i_k}\})$ for each subset $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. However, restructuring requires some care with labels. If we simply reused the label X in the last property in Definition 5, we would obtain

$$X\{X_1(X'_1) \oplus X_2(X'_2)\} \equiv X(X_1\{X'_1\}, X_2\{X'_2\}) \geq X(X_1\{X'_1\}) \geq X(X_1\{\lambda\}) \geq X\{\lambda\}.$$

However, the last step here is wrong, as the left hand side is an indicator for the subset containing the elements with label X_1 being empty or not, whereas the right hand side is the corresponding indicator for the whole set, i.e. elements with labels X_1 or X_2. No such mapping can be claimed. In fact, what we really have to do is to mark the set label in an attribute of the form $X\{X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)\}$ to indicate the inner union attribute, i.e. we should use $X_{\{1,\ldots,n\}}$ (or even $X_{\{X_1,\ldots,X_n\}}$) instead of X. As long as we are not dealing with subattributes of the form $X_{\{1,\ldots,k\}}\{\lambda\}$, the additional index does not add any information and thus can be omitted to increase readability. The same applies to the multiset- and the list-constructor.

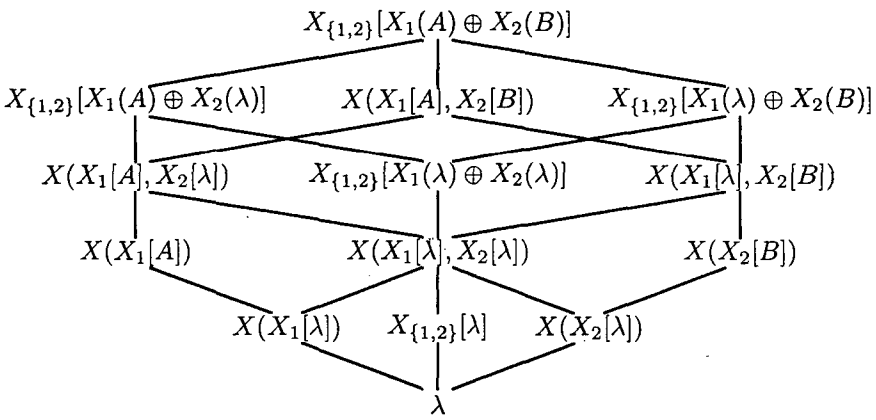


Figure 2: The lattice $\mathcal{S}(X[X_1(A) \oplus X_2(B)])$

Subattributes of the form $X_I\{\lambda\}$, $X_I[\lambda]$ and $X_I\{\lambda\}$ were called *counter attributes* in [27], because they can be considered as counters for the number of elements in a list or multiset or as flags that tell, whether sets are empty or not. Note that $X_\emptyset\{\lambda\} = \lambda$, $X_{\{1,\ldots,n\}}\{\lambda\} = X\{\lambda\}$ and $X_{\{i\}}\{\lambda\} = X(X_i\{\lambda\})$. Analogous conventions apply to list and multiset attributes.

Further note that due to the restructuring rules in Definitions 4 and 5 we may have the case that a record attribute is a subattribute of a set attribute and vice versa. This cannot be the case, if the union-constructor is absent. However, the presence of the restructuring rules allows us to assume that the union-constructor only appears inside a set-constructor or as the outermost constructor. This will be frequently exploited in our proofs.

Obviously, $X \geq Y$ induces a projection map $\pi_Y^X : dom(X) \to dom(Y)$. For $X \equiv Y$ we have $X \geq Y$ and $Y \geq X$ and the projection maps π_Y^X and π_X^Y are inverse to each other.

We use the notation $\mathcal{S}(X) = \{Z \in \mathcal{N} \mid X \geq Z\}$ to denote the *set of subattributes* of a nested attribute X. Figure 1 shows the subattributes of $X\{X_1(A) \oplus X_2(B)\} = X(X_1\{A\}, X_2\{B\})$ together with the relation $\geq$ on them. Note that the subattribute $X_{\{1,2\}}\{\lambda\}$ would not occur, if we only considered the record-structure, whereas other subattributes such as $X(X_i\{\lambda\})$ would not occur, if we only considered the set-structure. This is a direct consequence of the restructuring rules.

Figure 2 shows the subattributes of $X[X_1(A) \oplus X_2(B)]$ together with the relation $\geq$ on them. The subattributes $X_{\{1,2\}}[\lambda]$ would not occur, if we only considered the list-structure, whereas other subattributes such as $X(X_i[\lambda])$ would not occur, if we ignored the restructuring rules. Figure 3 shows the subattributes of $X\{X_1(A) \oplus X_2(B) \oplus X_3(C)\}$ together with the relation $\geq$ on them. The subattribute $X_I\{\lambda\}$ for $|I| \geq 2$ would not occur, if we only considered the record-structure.

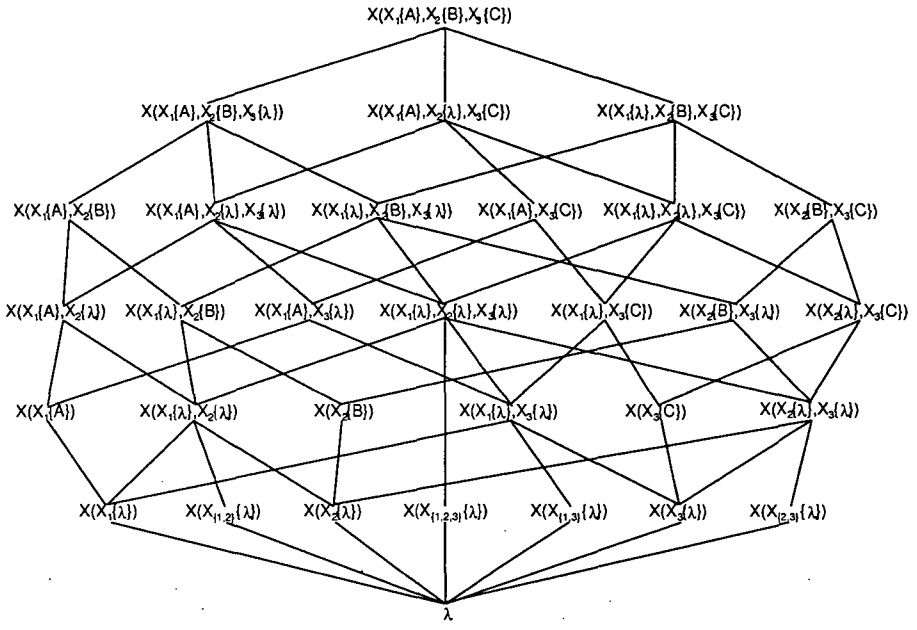


Figure 3: The subattribute lattice $\mathcal{S}(X\{X_1(A) \oplus X_2(B) \oplus X_3(C)\})$

2.3 The Lattice Structure

The set of subattributes $\mathcal{S}(X)$ of a nested attribute X plays the same role in the dependency theory for higher-order data models as the powerset $\mathcal{P}(R)$ for a relation schema R plays in the dependency theory for the relational model. $\mathcal{P}(R)$ is a Boolean algebra with order $\subseteq$, intersection $\cap$, union $\cup$ and the difference $-$. So, the question arises which algebraic structure $\mathcal{S}(X)$ carries.

Definition 6. Let $\mathcal{L}$ be a lattice with zero and one, partial order $\leq$, join $\sqcup$ and meet $\sqcap$. $\mathcal{L}$ has *relative pseudo-complements* iff for all $Y, Z \in \mathcal{L}$ the infimum $Y \leftarrow Z = \bigsqcap\{U \mid U \sqcup Y \geq Z\}$ exists. Then $Y \leftarrow 1$ (1 being the one in $\mathcal{L}$) is called the *relative complement* of Y.

If we have distributivity in addition, we call $\mathcal{L}$ a *Brouwer algebra*. In this case the relative pseudo-complements satisfy $U \geq (Y \leftarrow Z)$ iff $(U \sqcup Y \geq Z)$, but if we do not have distributivity this property may be violated though relative pseudo-complements exist.

Theorem 1. *The set $\mathcal{S}(X)$ of subattributes carries the structure of a lattice with zero and one and relative pseudo-complements, where the order $\geq$ is as defined in Definition 5, and λ and X are the zero and one, respectively. If X does not contain the union constructor, $\mathcal{S}(X)$ defines a Brouwer algebra.*

Proof. For $X = \lambda$ and simple attributes $X = A$ we obtain trivial lattices with only one or two elements. Applying the record constructor leads to a cartesian product of lattices, while the set, list and multiset constructors add a new zero element to a lattice. These extensions preserve the properties of a Brouwer algebra.

In the case of set, list and multiset constructors applied to a union attribute we add counter attributes. This preserves the properties of a lattice and the existence of relative pseudo-complement, while distributivity may be lost.

□

Example 1. Let $X = X\{X_1(A) \oplus X_2(B)\}$ with $\mathcal{S}(X)$ as illustrated in Figure 1, $Y_1 = X\{\lambda\}$, $Y_2 = X(X_2\{B\})$, and $Z = X(X_1\{A\})$. Then we have

$$Z \sqcap (Y_1 \sqcup Y_2) = X(X_1\{A\}) \sqcap (X\{\lambda\} \sqcup X(X_2\{B\})) =$$
$$X(X_1\{A\}) \sqcap X(X_1\{\lambda\}, X_2\{B\}) = X(X_1\{\lambda\}) \neq \lambda = \lambda \sqcup \lambda =$$
$$(X(X_1\{A\}) \sqcap X\{\lambda\}) \sqcup (X(X_1\{A\}) \sqcap X(X_2\{B\})) = (Z \sqcap Y_1) \sqcup (Z \sqcap Y_2)\,.$$

This shows that $\mathcal{S}(X)$ in general is not a distributive lattice. Furthermore, $Y' \sqcup Z \geq Y_1$ holds for all Y' except λ, $X(X_1\{\lambda\})$ and $X(X_1\{A\})$. So $Z \leftarrow Y_1 = \lambda$, but not all $Y' \geq \lambda$ satisfy $Y' \sqcup Z \geq Y_1$.

It is easy to determine explicit inductive definitions of the operations $\sqcap$ (meet), $\sqcup$ (join) and $\leftarrow$ (relative pseudo-complement). This can be done by boring technical verification of the properties of meets, joins and relative pseudo-complements and is therefore omitted here.

3 Coincidence Ideals

In this section we investigate sets of subattributes, on which two complex values coincide. It is rather easy to see that these turn out to be ideals in the lattice $\mathcal{S}(X)$, i.e. they are non-empty and downward-closed. Therefore, we will call them *coincidence ideals.* However, there are many other properties that hold for coincidence ideals.

There are two major reasons for looking at coincidence ideals. The first one is that properties of subattributes, on which two complex values coincide, may give rise to axioms for functional dependencies. We will indeed see that the properties of coincidence ideals in Definition 7 are very closely related to the sound axioms and rules that we will derive in Theorems 3, 5 and 6.

The second reason is that in the completeness proof we will have to construct two complex values that coincide exactly on a given set of attributes, so that a set of dependencies is satisfied by these values, while a non-derivable dependency is not. This step appears also in the corresponding completeness proof for the RDM, but in that case it is trivial, because it simply amounts to getting two tuples that coincide on a given set of attributes, but differ on all others.

Thus, what we want to achieve is a characterisation of a coincidence ideal that allows us to construct two complex values that coincide exactly on it. This will be the main result of this section, called the Central Theorem 2 on coincidence ideals.

The proof of this result in [28] is very technical. In a nutshell, what we did was to discover properties of coincidence ideals, "translate" them into axioms for (weak) functional dependencies, ensure that we can rediscover these properties from the particular set of subattributes that arises naturally in the completeness proof (see Lemma 2), which required to weaken the axioms as much as possible, and finally show that the properties are sufficient for the desired Central Theorem.

Definition 7. A subset $\mathcal{F} \subseteq \mathcal{S}(X)$ is called a *coincidence ideal* on $\mathcal{S}(X)$ iff there exist complex values $t_1, t_2 \in dom(X)$ such that $\mathcal{F} = \{Y \in \mathcal{S}(X) \mid \pi_Y^X(t_1) = \pi_Y^X(t_2)\} \subseteq \mathcal{S}(X)$ is the set of subattributes, on which they coincide.

In [18] and in [26] the term "SHL-ideal" was used instead; in [19] in a restricted setting the term "HL-ideal" was used. Note that in all these cases not all the conditions in Theorem 2 were yet present.

In order to characterise sufficient and necessary properties of coincidence ideals we will need the notion of reconsilable subattributes, which was already used in the axiomatisations of restricted cases (see [19, 20]). The following Definition 8 extends this notion to capture all constructors, in particular the union constructor.

Definition 8. Two subattributes $Y, Z \in \mathcal{S}(X)$ are called *reconsilable* iff one of the following holds:

1. $Y \geq Z$ or $Z \geq Y$;
2. $X = X[X']$, $Y = X[Y']$, $Z = X[Z']$ and $Y', Z' \in \mathcal{S}(X')$ are reconsilable;
3. $X = X(X_1, \ldots, X_n)$, $Y = X(Y_1, \ldots, Y_n)$, $Z = X(Z_1, \ldots, Z_n)$ and $Y_i, Z_i \in \mathcal{S}(X_i)$ are reconsilable for all $i = 1, \ldots, n$;
4. $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$, $Y = X_1(Y_1') \oplus \cdots \oplus X_n(Y_n')$, $Z = X_1(Z_1') \oplus \cdots \oplus X_n(Z_n')$ and $Y_i', Z_i' \in \mathcal{S}(X_i')$ are reconsilable for all $i = 1, \ldots, n$;
5. $X = X[X_1(X_1') \oplus \cdots \oplus X_n(X_n')]$, $Y = X(Y_1, \ldots, Y_n)$ with $Y_i = X_i[Y_i']$ or $Y_i = \lambda = Y_i'$, $Z = X[X_1(Z_1') \oplus \cdots \oplus X_n(Z_n')]$, and Y_i', Z_i' are reconsilable for all $i = 1, \ldots, n$.

Note that for the set- and multiset-constructor we can only obtain reconsilability for subattributes in a $\geq$-relation.

Theorem 2 (Central Theorem). *Let $X \in \mathcal{N}$ be a nested attribute. Then $\mathcal{F} \subseteq \mathcal{S}(X)$ is a coincidence ideal iff the following conditions are satisfied:*

1. $\lambda \in \mathcal{F}$;
2. *if $Y \in \mathcal{F}$ and $Z \in \mathcal{S}(X)$ with $Y \geq Z$, then $Z \in \mathcal{F}$;*
3. *if $Y, Z \in \mathcal{F}$ are reconsilable, then $Y \sqcup Z \in \mathcal{F}$;*
4. *a) if $X_I\{\lambda\} \in \mathcal{F}$ and $X_J\{\lambda\} \notin \mathcal{F}$ for $I \subsetneq J$, then $X(X_{i_1}\{X_{i_1}'\}, \ldots, X_{i_k}\{X_{i_k}'\}) \in \mathcal{F}$ for $I = \{i_1, \ldots, i_k\}$;*

b) if $X_I\{\lambda\} \in \mathcal{F}$ *and* $X(X_i\{\lambda\}) \notin \mathcal{F}$ *for all* $i \in I$, *then there is a partition* $I = I_1 \cup I_2$ *with* $X_{I_1}\{\lambda\} \notin \mathcal{F}$, $X_{I_2}\{\lambda\} \notin \mathcal{F}$ *and* $X_{I'}\{\lambda\} \in \mathcal{F}$ *for all* $I' \subseteq I$ *with* $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$;

c) if $X_{\{1,\ldots,n\}}\{\lambda\} \in \mathcal{F}$ *and* $X_{I^-}\{\lambda\} \notin \mathcal{F}$ *(for* $I^- = \{i \in \{1,\ldots,n\} \mid X(X_i\{\lambda\}) \notin \mathcal{F}\}$*), then there exists some* $i \in I^+ = \{i \in \{1,\ldots,n\} \mid X(X_i\{\lambda\}) \in \mathcal{F}\}$ *such that for all* $J \subseteq I^-$ $X_{J\cup\{i\}}\{\lambda\} \in \mathcal{F}$ *holds;*

d) if $X_J\{\lambda\} \notin \mathcal{F}$ *and* $X_{\{j\}}\{\lambda\} \notin \mathcal{F}$ *for all* $j \in J$ *and for all* $i \in I$ *there is some* $J_i \subseteq J$ *with* $X_{J_i\cup\{i\}}\{\lambda\} \notin \mathcal{F}$, *then* $X_{I\cup J}\{\lambda\} \notin \mathcal{F}$, *provided* $I \cap J = \emptyset$;

e) if $X_{I^-}\{\lambda\} \in \mathcal{F}$ *and* $I' \subseteq I^+$ *such that for all* $i \in I'$ *there is some* $J \subseteq I^-$ *with* $X_{J\cup\{i\}}\{\lambda\} \notin \mathcal{F}$, *then* $X_{I'\cup J'}\{\lambda\} \notin \mathcal{F}$ *for all* $J' \subseteq I^-$ *with* $X_{J'}\{\lambda\} \notin \mathcal{F}$;

5. *a) if* $X_I\{\lambda\} \in \mathcal{F}$ *and* $X_J\{\lambda\} \in \mathcal{F}$ *with* $I \cap J = \emptyset$, *then* $X_{I\cup J}\{\lambda\} \in \mathcal{F}$;

b) if $X_I[\lambda] \in \mathcal{F}$ *and* $X_J[\lambda] \in \mathcal{F}$ *with* $I \cap J = \emptyset$, *then* $X_{I\cup J}[\lambda] \in \mathcal{F}$;

c) if $X_I\langle\lambda\rangle \in \mathcal{F}$ *and* $X_J\langle\lambda\rangle \in \mathcal{F}$ *with* $I \cap J = \emptyset$, *then* $X_{I\cup J}\langle\lambda\rangle \in \mathcal{F}$;

d) if $X_I[\lambda] \in \mathcal{F}$ *and* $X_J[\lambda] \in \mathcal{F}$ *with* $J \subseteq I$, *then* $X_{I-J}[\lambda] \in \mathcal{F}$;

e) if $X_I\langle\lambda\rangle \in \mathcal{F}$ *and* $X_J\langle\lambda\rangle \in \mathcal{F}$ *with* $J \subseteq I$, *then* $X_{I-J}\langle\lambda\rangle \in \mathcal{F}$;

f) if $X_I[\lambda] \in \mathcal{F}$ *and* $X_J[\lambda] \in \mathcal{F}$, *then* $X_{I\cap J}[\lambda] \in \mathcal{F}$ *iff* $X_{(I-J)\cup(J-I)}[\lambda] \in \mathcal{F}$;

g) if $X_I\langle\lambda\rangle \in \mathcal{F}$ *and* $X_J\langle\lambda\rangle \in \mathcal{F}$, *then* $X_{I\cap J}\langle\lambda\rangle \in \mathcal{F}$ *iff* $X_{(I-J)\cup(J-I)}\langle\lambda\rangle \in \mathcal{F}$;

6. *a) for* $X = X\{\bar{X}\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}\}$, *whenever* $I \subseteq \{1,\ldots,n\}$, *there is a partition* $I = I^- \cup I_{+-} \cup I_+ \cup I_-$ *such that*

   i. $X\{\bar{X}_{\{i\}}\{\lambda\}\} \in \mathcal{F}$ *iff* $i \notin I^-$,

   ii. $X\{\bar{X}_{I'}\{\lambda\}\} \in \mathcal{F}$, *whenever* $I' \cap I_+ \neq \emptyset$,

   iii. $X\{\bar{X}_{I'}\{\lambda\}\} \in \mathcal{F}$ *iff* $X\{\bar{X}_{I'\cap(I_{+-}\cup I^-)}\{\lambda\}\} \in \mathcal{F}$, *whenever* $I' \subseteq I_{+-} \cup I^- \cup I_-$;

b) for $X = X\langle\bar{X}\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}\rangle$, *whenever* $I \subseteq \{1,\ldots,n\}$, *there is a partition* $I = I^- \cup I_{+-} \cup I_+ \cup I_-$ *such that*

   i. $X\langle\bar{X}_{\{i\}}\{\lambda\}\rangle \in \mathcal{F}$ *iff* $i \notin I^-$,

   ii. $X\langle\bar{X}_{I'}\{\lambda\}\rangle \in \mathcal{F}$, *whenever* $I' \cap I_+ \neq \emptyset$,

   iii. $X\langle\bar{X}_{I'}\{\lambda\}\rangle \in \mathcal{F}$ *iff* $X\langle\bar{X}_{I'\cap(I_{+-}\cup I^-)}\{\lambda\}\rangle \in \mathcal{F}$, *whenever* $I' \subseteq I_{+-} \cup I^- \cup I_-$;

7. *a) if* $X = X(X_1',\ldots,X_n')$, *then* $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X_i') \mid X(\lambda,\ldots,Y_i,\ldots,\lambda) \in \mathcal{F}\}$ *is a coincidence ideal;*

b) if $X = X[X']$, *such that* X' *is not a union attribute, and* $\mathcal{F} \neq \{\lambda\}$, *then* $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X[Y] \in \mathcal{F}\}$ *is a coincidence ideal;*

c) If $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$ *and* $\mathcal{F} \neq \{\lambda\}$, *then the set* $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X_i') \mid X_1(\lambda) \oplus \cdots \oplus X_i(Y_i) \oplus \cdots \oplus X_n(\lambda) \in \mathcal{F}\}$ *is a coincidence ideal;*

d) if $X = X\{X'\}$, such that X' is not a union attribute, and $\mathcal{F} \neq \{\lambda\}$, then $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X\{Y\} \in \mathcal{F}\}$ is a defect coincidence ideal;

e) if $X = X\langle X'\rangle$, such that X' is not a union attribute, and $\mathcal{F} \neq \langle\lambda\rangle$, then $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X\langle Y\rangle \in \mathcal{F}\}$ is a defect coincidence ideal.

In property 7 of the theorem a *defect coincidence ideal* on $\mathcal{S}(X)$ is a subset $\mathcal{F} \subseteq \mathcal{S}(X)$ satisfying properties 1, 2, 4(a)-(d), 6(a),(b), 7(d)-(e) and

8. a) if $X = X(X'_1, \ldots, X'_n)$, then $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X'_i) \mid X(\lambda, \ldots, Y_i, \ldots, \lambda) \in \mathcal{F}\}$ is a defect coincidence ideal;

   b) if $X = X[X']$, such that X' is not a union attribute, and $\mathcal{F} \neq \{\lambda\}$, then $\mathcal{G} = \{Y \in \mathcal{S}(X') \mid X[Y] \in \mathcal{F}\}$ is a defect coincidence ideal;

   c) If $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ and $\mathcal{F} \neq \{\lambda\}$, then the set $\mathcal{F}_i = \{Y_i \in \mathcal{S}(X'_i) \mid X_1(\lambda) \oplus \cdots \oplus X_i(Y_i) \oplus \cdots \oplus X_n(\lambda) \in \mathcal{F}\}$ is a defect coincidence ideal.

The proof of Theorem 2, in particular, showing that the conditions are sufficient, is very technical and lengthy (see [28]). The general idea is to use structural induction extending the corresponding proofs in [19] and in [20]. However, a difficulty arises with the set and multiset constructors, as for them defect coincidence ideals have to be dealt with. The work in [20, Lemmata 21 and 24] contains a proof for the case that the union constructor does not appear at all. This has been generalised in [27, Lemma 4.3] to the general case but excluding counter attributes, i.e. attributes of the form $X_I\{\lambda\}$, $X_I\langle\lambda\rangle$ or $X_I[\lambda]$ with $|I| \geq 2$.

4 Functional Dependencies and Weak Functional Dependencies

In this section we will define functional and weak functional dependencies on $\mathcal{S}(X)$ and derive a sound and complete system of derivation rules for wFDs.

Definition 9. Let $X \in \mathcal{N}$. A *functional dependency* (FD) on $\mathcal{S}(X)$ is an expression $\mathcal{Y} \to \mathcal{Z}$ with $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X)$. A *weak functional dependency* (wFD) on $\mathcal{S}(X)$ is an expression $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with an index set I and $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X)$.

In the following we consider finite sets $r \subseteq dom(X)$, which we will call simply *instances* of X.

Definition 10. Let r be an instance of X. We say that r *satisfies the FD* $\mathcal{Y} \to \mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models \mathcal{Y} \to \mathcal{Z}$) iff for all $t_1, t_2 \in r$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ we also have $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$.

An instance $r \subseteq dom(X)$ *satisfies the wFD* $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ on $\mathcal{S}(X)$ (notation: $r \models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$) iff for all $t_1, t_2 \in r$ there is some $i \in I$ with $\{t_1, t_2\} \models \mathcal{Y}_i \to \mathcal{Z}_i$.

According to this definition we identify a wFD $\{\!|\mathcal{Y} \to \mathcal{Z}|\!\}$, i.e. the index set contains exactly one element, with the "ordinary" FD $\mathcal{Y} \to \mathcal{Z}$.

Note that our notion of weak functional dependencies is indeed more general than the one used in [32, p.75] based on the work by Demetrovics and Gyepesi (see [11]). The straighforward generalisation of the dependencies introduced by Demetrovics and Gyepesi would only lead to wFDs of the form $\{\!|\mathcal{Y} \to \{Z_i\} \mid i \in I|\!\}$, i.e. the left hand side of all involved FDs is always the same, while the right hand side only contains a single subattribute. Our notion of wFDs covers also so called *dual functional dependencies* (dFDs) (see [11]), which would take the form $\{\!|\{Y_i\} \to \{Z_j\} \mid i \in I, j \in J|\!\}$.

Let Σ be a set of FDs and wFDs. A FD or wFD ψ is implied by Σ (notation: $\Sigma \models \psi$) iff all instances r with $r \models \varphi$ for all $\varphi \in \Sigma$ also satisfy ψ. As usual we write $\Sigma^* = \{\psi \mid \Sigma \models \psi\}$.

As usual we write Σ^+ for the set of all FDs and wFDs that can be derived from Σ by applying a system $\mathfrak{R}$ of axioms and rules, i.e. $\Sigma^+ = \{\psi \mid \Sigma \vdash_{\mathfrak{R}} \psi\}$. We omit the standard definitions of derivations with a given rule system, and also write simply $\vdash$ instead of $\vdash_{\mathfrak{R}}$, if the rule system is clear from the context.

Our goal is to find a finite axiomatisation, i.e. a finite rule system $\mathfrak{R}$ such that $\Sigma^* = \Sigma^+$ holds. The rules in $\mathfrak{R}$ are *sound* iff $\Sigma^+ \subseteq \Sigma^*$ holds, and *complete* iff $\Sigma^* \subseteq \Sigma^+$ holds.

4.1 Sound Derivation Rules

Let us first look only at FDs. In general, two complex values in $dom(X)$ that coincide on subattributes Y and Z of X need not coincide on $Y \sqcup Z$. So we cannot expect a simple generalisation of Armstrong's extension rule for FDs in the relational model. However, the notion of "reconsilability" introduced in Definition 8 will permit such a generalisation.

Theorem 3. *The following axioms and rules are sound for the implication of FDs on $\mathcal{S}(X)$:*

reflexivity axiom:

$$\frac{}{\mathcal{Y} \to \mathcal{Z}}\ \mathcal{Z} \subseteq \mathcal{Y} \tag{1}$$

subattribute axiom:

$$\frac{}{\{Y\} \to \{Z\}}\ Y \geq Z \tag{2}$$

join axiom:

$$\frac{}{\{Y, Z\} \to \{Y \sqcup Z\}}\ Y, Z \text{ reconsilable} \tag{3}$$

λ axiom:

$$\frac{}{\emptyset \to \{\lambda\}} \tag{4}$$

extension rule:

$$\frac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \mathcal{Y} \cup \mathcal{Z}} \tag{5}$$

transitivity rule:

$$\frac{\mathcal{Y} \to \mathcal{Z} \quad \mathcal{Z} \to \mathcal{U}}{\mathcal{Y} \to \mathcal{U}} \tag{6}$$

Proof. The soundness of the axioms (1), (2) and (4) is trivial.

For (3) let $t_1, t_2 \in r$ for some instance $r \subseteq dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ and $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for reconsilable subattributes $Y, Z \in \mathcal{S}(X)$.

1. In case $Y \geq Z$ we have $Y \sqcup Z = Y$ and thus $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$.

2. In case $X = X[X']$ we must have $Y = X[Y']$ and $Z = X[Z']$ with reconsilable subattributes $Y', Z' \in \mathcal{S}(X')$. Furthermore, $t_1 = [t_{1,1}, \ldots, t_{1,n}]$ and $t_2 = [t_{2,1}, \ldots, t_{2,m}]$. This gives $n = m$, $\pi^{X'}_{Y'}(t_{1,j}) = \pi^{X'}_{Y'}(t_{2,j})$ and $\pi^{X'}_{Z'}(t_{1,j}) = \pi^{X'}_{Z'}(t_{2,j})$ for all $j = 1, \ldots, n$. By induction we obtain $\pi^{X'}_{Y' \sqcup Z'}(t_{1,j}) = \pi^{X'}_{Y' \sqcup Z'}(t_{2,j})$ for all $j = 1, \ldots, n$. From this and $Y \sqcup Z = X[Y' \sqcup Z']$ follows $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$.

3. In case $X = X(X_1, \ldots, X_n)$ we must have $Y = X(Y_1, \ldots, Y_n)$ and $Z = X(Z_1, \ldots, Z_n)$ with reconsilable subattributes $Y_i, Z_i \in \mathcal{S}(X_i)$ for $i = 1, \ldots, n$. Furthermore, $t_1 = (t_{1,1}, \ldots, t_{1,n})$ and $t_2 = (t_{2,1}, \ldots, t_{2,n})$, which implies $\pi^{X_i}_{Y_i}(t_{1,i}) = \pi^{X_i}_{Y_i}(t_{2,i})$ and $\pi^{X_i}_{Z_i}(t_{1,i}) = \pi^{X_i}_{Z_i}(t_{2,i})$ for all $i = 1, \ldots, n$. By induction we obtain $\pi^{X_i}_{Y_i \sqcup Z_i}(t_{1,i}) = \pi^{X_i}_{Y_i \sqcup Z_i}(t_{2,i})$ for all $i = 1, \ldots, n$. From this and $Y \sqcup Z = X(Y_1 \sqcup Z_1, \ldots, Y_n \sqcup Z_n)$ follows $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$.

4. In case $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$ we must have $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ and $Z = X_1(Z_1) \oplus \cdots \oplus X_n(Z_n)$ with reconsilable subattributes $Y_i, Z_i \in \mathcal{S}(X'_i)$ for $i = 1, \ldots, n$. Furthermore $t_1 = (X_i : t'_1)$ and $t_2 = (X_i : t'_2)$ for some $i \in \{1, \ldots, n\}$, which implies $\pi^{X'_i}_{Y_i}(t'_1) = \pi^{X'_i}_{Y_i}(t'_2)$ and $\pi^{X'_i}_{Z_i}(t'_1) = \pi^{X'_i}_{Z_i}(t'_2)$. By induction we obtain $\pi^{X'_i}_{Y_i \sqcup Z_i}(t'_1) = \pi^{X'_i}_{Y_i \sqcup Z_i}(t'_2)$. Finally, $Y \sqcup Z = X_1(Y_1 \sqcup Z_1) \oplus \cdots \oplus X_n(Y_n \sqcup Z_n)$ implies $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$ as desired.

5. In case $X = X[X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)]$ we must have $Y = X(Y_1, \ldots, Y_n)$ with $Y_i = X_i[Y'_i]$ or $Y_i = \lambda = Y'_i$, and $Z = X[X_1(Z'_1) \oplus \cdots \oplus X_n(Z'_n)]$, such that Y'_i, Z'_i are reconsilable for all $i = 1, \ldots, n$. We get $Y \sqcup Z = X[X_1(Y'_1 \sqcup Z'_1) \oplus \cdots \oplus X_n(Y'_n \sqcup Z'_n)]$. As $Z \geq X[\lambda]$, we also have $\pi^X_{X[\lambda]}(t_1) = \pi^X_{X[\lambda]}(t_2)$, so t_1 and t_2 are lists of equal length. Therefore, assume $t_j = [t_{j1}, \ldots, t_{jm}]$ for $j = 1, 2$ and $t_{jk} = (X_\ell : t''_{jk})$. This gives $\pi^X_{Y \sqcup Z}(t_j) = [t'_{j1}, \ldots, t'_{jm}]$ with $t'_{jk} = (X_\ell : \pi^{X'_\ell}_{Y'_\ell \sqcup Z'_\ell}(t''_{jk}))$. We know $\pi^{X'_\ell}_{Z'_\ell}(t''_{1k}) = \pi^{X'_\ell}_{Z'_\ell}(t''_{2k})$, so we are done for $Y_\ell = \lambda$. For $Y_\ell \neq \lambda$ the sublists containing all $(X_\ell : t''_{jk})$ coincide on Y'_ℓ. As Y'_ℓ and Z'_ℓ are semi-disjoint, we have $\pi^{X'_\ell}_{Y'_\ell \sqcup Z'_\ell}(t''_{1k}) = \pi^{X'_\ell}_{Y'_\ell \sqcup Z'_\ell}(t''_{2k})$ by induction, which implies $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$.

For the extension rule (5) let $t_1, t_2 \in r$ for some instance $r \subseteq dom(X)$ with $r \models \mathcal{Y} \to \mathcal{Z}$, and assume $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ holds for all $Y \in \mathcal{Y}$. Then we must have as well $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$, which implies $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y} \cup \mathcal{Z}$, i.e. $r \models \mathcal{Y} \to \mathcal{Y} \cup \mathcal{Z}$.

For the transitivity rule (6) let $t_1, t_2 \in r$ for some instance $r \subseteq dom(X)$ with $r \models \mathcal{Y} \to \mathcal{Z}$ and $r \models \mathcal{Z} \to \mathcal{U}$, and assume $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ holds for all $Y \in \mathcal{Y}$. Then we must have as well $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$ by the first premise, and hence $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}$ by the second premise, which shows $r \models \mathcal{Y} \to \mathcal{U}$ as desired.

□

In [20] it was shown that the six of axioms and rules in Theorem 3, i.e. (1) – (6) are complete for the implication of FDs, if the union constructor is not present. In this case (2), (3) and (4) are axioms that deal with the Brouwer algebra structure on $\mathcal{S}(X)$, while (1), (5) and (6) are the well known Armstrong axioms and rules.

Theorem 4. *The following rules for the implication of FDs on $\mathcal{S}(X)$ can be derived from the rules in Theorem 3:*

union rule:

$$\frac{\mathcal{Y} \to \mathcal{Z} \quad \mathcal{Y} \to \mathcal{U}}{\mathcal{Y} \to \mathcal{Z} \cup \mathcal{U}} \tag{7}$$

fragmentation rule:

$$\frac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \{Z\}}\ Z \in \mathcal{Z} \tag{8}$$

join rule:

$$\frac{\{Y\} \to \{Z\}}{\{Y\} \to \{Y \sqcup Z\}}\ Y, Z \text{ reconsilable} \tag{9}$$

Proof. For the union rule (7) we use the following derivation:

$$\dfrac{\dfrac{\mathcal{Y} \to \mathcal{Z}}{\mathcal{Y} \to \mathcal{Y} \cup \mathcal{Z}} \quad \dfrac{\dfrac{\dfrac{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Y} \quad \mathcal{Y} \to \mathcal{U}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{U}}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Y} \cup \mathcal{Z} \cup \mathcal{U}} \quad \dfrac{}{\mathcal{Y} \cup \mathcal{Z} \cup \mathcal{U} \to \mathcal{Z} \cup \mathcal{U}}}{\mathcal{Y} \cup \mathcal{Z} \to \mathcal{Z} \cup \mathcal{U}}}{\mathcal{Y} \to \mathcal{Z} \cup \mathcal{U}}$$

For the fragmentation rule (8) we use the following derivation:

$$\frac{\mathcal{Y} \to \mathcal{Z} \quad \dfrac{}{\mathcal{Z} \to \{Z\}}}{\mathcal{Y} \to \{Z\}}$$

Finally, for the join-rule (9) we use the following derivation:

$$\dfrac{\dfrac{\{Y\} \to \{Z\}}{\{Y\} \to \{Y, Z\}} \quad \dfrac{}{\{Y, Z\} \to \{Y \sqcup Z\}}}{\{Y\} \to \{Y \sqcup Z\}}$$

□

If the union constructor is present, we obtain further subattributes, for which we obtain additional axioms. These will be set, multiset and list axioms (10) - (18) in the following Theorem 5. Furthermore, we obtain rules that derive FDs on $\mathcal{S}(X)$ from FDs on $\mathcal{S}(X')$ for *embedded attributes* X', i.e. X' results from X by stripping away the outermost constructor. The following definition clarifies in an exact way, how embedded attributes and induced instances for embedded attributes have to be understood. This will become important also for the extensions in Section 5.

Definition 11. Let $X \in \mathcal{N}$ be a nested attribute. The *set of embedded attributes* of X is the smallest set $\mathrm{emb}(X)$ with $X \in \mathrm{emb}(X)$ satisfying the following properties:

1. If $X = X(X_1, \ldots, X_n)$ is a record attribute, then $\mathrm{emb}(X_i) \subseteq \mathrm{emb}(X)$ holds for all $i = 1, \ldots, n$.
2. If $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$ is a union attribute, then $\mathrm{emb}(X_i') \subseteq \mathrm{emb}(X)$ holds for all $i = 1, \ldots, n$.
3. If $X = X\{X'\}$ is a set attribute, then $\mathrm{emb}(X') \subseteq \mathrm{emb}(X)$ holds.
4. If $X = X[X']$ is a list attribute, then $\mathrm{emb}(X') \subseteq \mathrm{emb}(X)$ holds.
5. If $X = X\langle X'\rangle$ is a multiset attribute, then $\mathrm{emb}(X') \subseteq \mathrm{emb}(X)$ holds.

If $r \subseteq dom(X)$ is an instance of X, then for each $Y \in \mathrm{emb}(X)$ we obtain the *induced instance* $r \downarrow Y$ in the following way:

1. $r \downarrow X = r$;
2. $r \downarrow Z = (r \downarrow Y) \downarrow Z$ for $Z \in \mathrm{emb}(Y)$ and $Y \in \mathrm{emb}(X)$;
3. $r \downarrow X_i = \{t_i \in dom(X_i) \mid \exists t \in r.t = (t_1, \ldots, t_i, \ldots, t_n)\}$ for a record attribute $X = X(X_1, \ldots, X_n)$;
4. $r \downarrow X_i = \{t_i \in dom(X_i) \mid \exists t \in r.t = (X_i : t_i)\}$ for a union attribute $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$;
5. $r \downarrow X' = \{t' \in dom(X') \mid \exists t \in r.t' \in t\}$ for a set attribute $X = X\{X'\}$;
6. $r \downarrow X' = \{t' \in dom(X') \mid \exists t \in r.t' \in t\}$ for a multiset attribute $X = X\langle X'\rangle$;
7. $r \downarrow X' = \{t' \in dom(X') \mid \exists t \in r.t = [\ldots, t', \ldots]\}$ for a list attribute $X = X[X']$.

In dealing now with FDs $\mathcal{Y} \to \mathcal{Z}$ defined embedded attributes $U \in \mathrm{emb}(X)$ we let $r \models \mathcal{Y} \to \mathcal{Z}$ mean $r \downarrow U \models \mathcal{Y} \to \mathcal{Z}$. This generalises canonically to wFDs.

Theorem 5. *In addition to the axioms and rules in Theorem 3 the following axioms and rules are sound for the implication of FDs on* $\mathcal{S}(X)$*:*

set axiom:

$$\frac{}{\{X_I\{\lambda\}, X_J\{\lambda\}\} \to \{X_{I\cup J}\{\lambda\}\}}\, I \cap J = \emptyset \tag{10}$$

multiset axioms:

$$\frac{}{\{X_I\langle\lambda\rangle, X_J\langle\lambda\rangle\} \to \{X_{I\cup J}\langle\lambda\rangle\}}\, I \cap J = \emptyset \tag{11}$$

$$\frac{}{\{X_I\langle\lambda\rangle, X_{I\cup J}\langle\lambda\rangle\} \to \{X_J\langle\lambda\rangle\}}\, I \cap J = \emptyset \tag{12}$$

$$\frac{}{\{X_I\langle\lambda\rangle, X_J\langle\lambda\rangle, X_{I\cap J}\langle\lambda\rangle\} \to \{X_{(I-J)\cup(J-I)}\langle\lambda\rangle\}} \tag{13}$$

$$\frac{}{\{X_I\langle\lambda\rangle, X_J\langle\lambda\rangle, X_{(I-J)\cup(J-I)}\langle\lambda\rangle\} \to \{X_{I\cap J}\langle\lambda\rangle\}} \tag{14}$$

list axioms:

$$\frac{}{\{X_I[\lambda], X_J[\lambda]\} \to \{X_{I\cup J}[\lambda]\}}\, I \cap J = \emptyset \tag{15}$$

$$\frac{}{\{X_I[\lambda], X_{I\cup J}[\lambda]\} \to \{X_J[\lambda]\}}\, I \cap J = \emptyset \tag{16}$$

$$\frac{}{\{X_I[\lambda], X_J[\lambda], X_{I\cap J}[\lambda]\} \to \{X_{(I-J)\cup(J-I)}[\lambda]\}} \tag{17}$$

$$\frac{}{\{X_I[\lambda], X_J[\lambda], X_{(I-J)\cup(J-I)}[\lambda]\} \to \{X_{I\cap J}[\lambda]\}} \tag{18}$$

set lifting rule:

$$\frac{\{Y\} \to \mathcal{Z}}{\{X\{Y\}\} \to \{X\{Z\} \mid Z \in \mathcal{Z}\}}\, X = X\{X'\},\ Y \in \mathcal{S}(X'), \mathcal{Z} \subseteq \mathcal{S}(X') \tag{19}$$

record lifting rule:

$$\frac{\mathcal{Y}_i \to \mathcal{Z}_i}{\{X(\lambda, \ldots, Y_i, \ldots, \lambda) \mid Y_i \in \mathcal{Y}_i\} \to \{X(\lambda, \ldots, Z_i, \ldots, \lambda) \mid Z_i \in \mathcal{Z}_i\}}\, \mathcal{C} \tag{20}$$

with conditions $\mathcal{C} : X = X(X_1, \ldots, X_n)$ *and* $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X_i)$

union lifting rule:

$$\frac{\mathcal{Y}_i \to \mathcal{Z}_i}{\{\cdots \oplus X_i(Y_i) \oplus \cdots \mid Y_i \in \mathcal{Y}_i\} \to \{\cdots \oplus X_i(Z_i) \oplus \cdots \mid Z_i \in \mathcal{Z}_i\}}\, \mathcal{C} \tag{21}$$

with conditions $\mathcal{C} : X = X(X_1, \ldots, X_n)$ *and* $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X_i'), \mathcal{Y}_i \neq \emptyset$

multiset lifting rule:

$$\frac{\{Y\} \to \mathcal{Z}}{\{X\langle Y\rangle\} \to \{X\langle Z\rangle \mid Z \in \mathcal{Z}\}} X = X\langle X'\rangle,\ Y \in \mathcal{S}(X'), \mathcal{Z} \subseteq \mathcal{S}(X') \qquad (22)$$

list lifting rule:

$$\frac{\mathcal{Y} \to \mathcal{Z}}{\{X[Y] \mid Y \in \mathcal{Y}\} \to \{X[Z] \mid Z \in \mathcal{Z}\}} X = X[X'],\ \mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X'), \mathcal{Y} \neq \emptyset \qquad (23)$$

Proof. For the set axiom (10) let $t_1, t_2 \in dom(X)$ with $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_I\{\lambda\}}(t_2)$ and $\pi^X_{X_J\{\lambda\}}(t_1) = \pi^X_{X_J\{\lambda\}}(t_2)$. In case $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_J\{\lambda\}}(t_1) = \emptyset$ there are no values of the form $(X_i : v_i)$ with $i \in I \cup J$ in t_1, hence also not in t_2. In case at least one of these projections leads to a non-empty set we must have $(X_i : v_i) \in t_1$ for at least one $i \in I \cup J$ and one value $v_i \in dom(X_i')$. The same holds for t_2, hence in both cases $\pi^X_{X_{I\cup J}\{\lambda\}}(t_1) = \pi^X_{X_{I\cup J}\{\lambda\}}(t_2)$.

For the first list axiom (15) let $t_1, t_2 \in dom(X)$. Then $\pi^X_{X_I[\lambda]}(t_1) = \pi^X_{X_I[\lambda]}(t_2)$ means that t_1 and t_2 contain the same number of elements of the form $(X_i : v_i)$ with $i \in I$. If the same holds for $I \cup J$, then t_1 and t_2 must also contain the same number of elements of the form $(X_i : v_i)$ with $i \in J$, i.e. $\pi^X_{X_J[\lambda]}(t_1) = \pi^X_{X_J[\lambda]}(t_2)$. The soundness of the second list axiom (16) follows from the same argument.

Analogously, for the third list axiom (17) for $Y \in \{X_I[\lambda], X_J[\lambda], X_{I\cap J}[\lambda]\}$ $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ means that t_1, t_2 contain the same number of elements with labels in I, J and $I \cap J$, respectively. So they also contain the same number of elements with labels in $(I - J) \cup (J - I)$. The soundness of the fourth list axiom (18) follows from the same argument.

The proof for the four multiset axioms (11) – (14) is completely analogous to the proof for the list axioms.

For the set lifting rule (19) let $t_1, t_2 \in dom(X)$ with $\pi^X_{X\{Y\}}(t_1) = \pi^X_{X\{Y\}}(t_2)$. Without loss of generality – repeat elements, if necessary – we may write $t_i = \{t_{i1}, \ldots, t_{ik}\}$ $(i = 1, 2)$. Then for all $j = 1, \ldots, k$ we have $\pi^{X'}_Y(t_{1j}) = \pi^{X'}_Y(t_{2j})$. From the premise of the rule we get $\pi^{X'}_Z(t_{1j}) = \pi^{X'}_Z(t_{2j})$ for all $j = 1, \ldots, k$ and all $Z \in \mathcal{Z}$, which implies $\pi^X_{X\{Z\}}(t_1) = \pi^X_{X\{Z\}}(t_2)$ for all $X\{Z\}$ with $Z \in \mathcal{Z}$.

For the record lifting rule (20) let $t_1, t_2 \in dom(X)$ with $\pi^X_{X(\lambda,\ldots,Y_i,\ldots,\lambda)}(t_1) = \pi^X_{X(\lambda,\ldots,Y_i,\ldots,\lambda)}(t_2)$ for all $Y_i \in \mathcal{Y}_i$. If $t_j = (t_{1j}, \ldots, t_{nj})$ for $j = 1, 2$, then it follows $\pi^{X_i}_{Y_i}(t_{i1}) = \pi^{X_i}_{Y_i}(t_{i2})$ for all $Y_i \in \mathcal{Y}_i$ and thus also $\pi^{X_i}_{Z_i}(t_{i1}) = \pi^{X_i}_{Z_i}(t_{i2})$ for all $Z_i \in \mathcal{Z}_i$ by the premise of the rule. This gives $\pi^X_{X(\lambda,\ldots,Z_i,\ldots,\lambda)}(t_1) = \pi^X_{X(\lambda,\ldots,Z_i,\ldots,\lambda)}(t_2)$ for all $Z_i \in \mathcal{Z}_i$ as desired.

For the union lifting rule (21) let $t_1, t_2 \in dom(X)$ with $\pi^X_{\ldots\oplus X_i(Y_i)\oplus\ldots}(t_1) = \pi^X_{\ldots\oplus X_i(Y_i)\oplus\ldots}(t_2)$ for all $Y_i \in \mathcal{Y}_i$. In particular, t_1 and t_2 must have the same label, and we can assume $t_j = (X_i : t_j')$ for $j = 1, 2$. Then we get $\pi^{X_i}_{Y_i}(t_1') = \pi^{X_i}_{Y_i}(t_2')$ for all $Y_i \in \mathcal{Y}_i$ and thus also $\pi^{X_i}_{Z_i}(t_1') = \pi^{X_i}_{Z_i}(t_2')$ for all $Z_i \in \mathcal{Z}_i$ by the premise of the rule. This implies $\pi^X_{\ldots\oplus X_i(Z_i)\oplus\ldots}(t_1) = \pi^X_{\ldots\oplus X_i(Z_i)\oplus\ldots}(t_2)$ for all $Z_i \in \mathcal{Z}_i$ as desired.

For the multiset lifting rule (22) let $t_1, t_2 \in dom(X)$ with $\pi^X_{X\langle Y\rangle}(t_1) = \pi^X_{X\langle Y\rangle}(t_2)$. In particular, t_1 and t_2 must contain the same number of elements, so we may write $t_i = \langle t_{i1}, \ldots, t_{ik}\rangle$ $(i = 1, 2)$. Then for all $j = 1, \ldots, k$ we obtain $\pi^{X'}_Y(t_{1j}) = \pi^{X'}_Y(t_{2j})$. From the premise of the rule we get $\pi^{X'}_Z(t_{1j}) = \pi^{X'}_Z(t_{2j})$ for all $j = 1, \ldots, k$ and all $Z \in \mathcal{Z}$, which implies $\pi^X_{X\langle Z\rangle}(t_1) = \pi^X_{X\langle Z\rangle}(t_2)$ for all $X\langle Z\rangle$ with $Z \in \mathcal{Z}$.

For the list lifting rule (23) let $t_1, t_2 \in dom(X)$ with $\pi^X_{X[Y]}(t_1) = \pi^X_{X[Y]}(t_2)$ for all $X[Y]$ with $Y \in \mathcal{Y}$. As $\mathcal{Y} \neq \emptyset$, it follows that t_1 and t_2 must have the same length, say $t_i = [t_{i1}, \ldots, t_{ik}]$ $(i = 1, 2)$, and for all $j = 1, \ldots, k$ and all $Y \in \mathcal{Y}$ we have $\pi^{X'}_Y(t_{1j}) = \pi^{X'}_Y(t_{2j})$. Hence $\pi^{X'}_Z(t_{1j}) = \pi^{X'}_Z(t_{2j})$ for all $j = 1, \ldots, k$ and all $Z \in \mathcal{Z}$, which implies $\pi^X_{X[Z]}(t_1) = \pi^X_{X[Z]}(t_2)$ for all $X[Z]$ with $Z \in \mathcal{Z}$.

□

According to the observation made before we may still say that all axioms and rules in Theorem 5 arise from the lattice structure on $\mathcal{S}(X)$.

The axioms and rules in Theorem 3 only apply to "ordinary" FDs. For the implication of wFDs we need additional axioms and rules.

Theorem 6. *The following axioms and rules are sound for the implication of wFDs on* $\mathcal{S}(X)$*:*

weakening rule:

$$\frac{\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}}{\{\!\{\mathcal{Y}_j \to \mathcal{Z}_j \mid j \in J\}\!\}}\ I \subseteq J \tag{24}$$

left union rule:

$$\frac{\{\!\{\mathcal{Y} \to \mathcal{Z}_i \mid i \in I\}\!\}}{\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}}\ \mathcal{Y} = \bigcup_{i \in I} \mathcal{Y}_i \tag{25}$$

shift rule:

$$\frac{\{\!\{\mathcal{Y} \cup \mathcal{U}_1 \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}_1)\}\!\} \ldots \{\!\{\mathcal{Y} \cup \mathcal{U}_k \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}_k)\}\!\}}{\{\!\{\mathcal{Y} \to \{Z\}, \mid Z \in \mathcal{Z}\}\!\}}\ \mathcal{C} \tag{26}$$

with condition $\mathcal{C}$: $\mathcal{P}(\mathcal{U}) = \{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$

union axiom *for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$ *and* $I = \{i_1, \ldots, i_k\}$*:*

$$\frac{}{\{\!\{X_I\{\lambda\}\} \to \{X_J\{\lambda\}\}, \{X_I\{\lambda\}\} \to \{X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})\}\}\!\}}\ I \subsetneq J \tag{27}$$

partition axiom *for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$ *and* $I \subseteq \{1, \ldots, n\}$*:*

$$\frac{}{\begin{array}{l}\{\!\{X_I\{\lambda\}\} \to \{X_{I_1' \cup I_2'}\{\lambda\} \mid \emptyset \neq I_1' \subseteq I_1, \emptyset \neq I_2' \subseteq I_2\},\\ \quad \{X_I\{\lambda\}\} \to \{X(X_i\{\lambda\})\} \mid I = I_1 \cup I_2, I_1 \cap I_2 = \emptyset, I_1 \neq \emptyset \neq I_2, i \in I\}\!\}\end{array}} \tag{28}$$

first plus/minus axiom *for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$*:*

$$\frac{}{\begin{array}{l}\{\!|\{\lambda\} \to \{X_{J\cup\{i\}}\{\lambda\} \mid J \subseteq I^-\}, \{X_{\{1,\ldots,n\}}\{\lambda\}\} \to \{X_{I^-}\{\lambda\}\},\\ \{X(X_j\{\lambda\})\} \to \{X\}, \{\lambda\} \to \{X(X_k\{\lambda\})\} \mid i,j \in I^+, k \in I^-|\!\}\end{array}}\ \mathcal{C} \qquad (29)$$

with condition $\mathcal{C} : \{1,\ldots,n\} = I^+ \dot\cup I^-$

second plus/minus axiom *for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$*:*

$$\frac{}{\begin{array}{l}\{\!|\{X_{I\cup J}\{\lambda\}\} \to \{X_J\{\lambda\}\},\ \{X_{I\cup J}\{\lambda\}\} \to \{X_{\{j\}}\{\lambda\}\},\\ \{X_{I\cup J}\{\lambda\}\} \to \{X_{J'\cup\{i_0\}}\{\lambda\} \mid J' \subseteq J\} \mid i_0 \in I, j \in J|\!\}\end{array}} \qquad (30)$$

with condition $I \cap J = \emptyset$

third plus/minus axiom *for* $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}$*:*

$$\frac{}{\begin{array}{l}\{\!|\{X_{I^-}\{\lambda\}, X_{I'\cup J'}\{\lambda\}, X_{\{i\}}\{\lambda\} \mid i \in I^+\} \to \{X_{J'}\{\lambda\}\},\\ \{X_{I^-}\{\lambda\}, X_{I'\cup J'}\{\lambda\}, X_{\{i\}}\{\lambda\} \mid i \in I^+\} \to \{X_{J\cup\{\ell\}}\{\lambda\} \mid J \subseteq I^-\},\\ \{X_{I^-}\{\lambda\}, X_{I'\cup J'}\{\lambda\}, X_{\{i\}}\{\lambda\} \mid i \in I^+\} \to \{X_{\{k\}}\{\lambda\}\} \mid k \in I^-, \ell \in I'|\!\}\end{array}} \qquad (31)$$

with conditions $I^+ \dot\cup I^- = \{1,\ldots,n\}$, $I^+ \cap I^- = \emptyset$, $I' \subseteq I^+$, $J' \subseteq I^-$

partition axiom for sets *for* $X = X\{\bar X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}\}$ *and* $P \subseteq \mathcal{P}(I)$*:*

$$\frac{}{\begin{array}{l}\{\!|\{\lambda\} \to \{X\{\bar X_{I'}\{\lambda\}\} \mid I' \cap I^+ \neq \emptyset\} \cup\\ \{X\{\bar X_{J_\cup J_-}\{\lambda\}\}, X\{\bar X_J\{\lambda\}\} \mid J_- \subseteq I_-, J \in Q, J \subseteq I_{+-} \dot\cup I^-\},\\ \{\lambda\} \to \{X\{\bar X_K\{\lambda\}\}\}, \{X\{\bar X_{K'}\{\lambda\}\}\} \to \{X\} \mid\\ I = I^- \dot\cup I_+ \dot\cup I_- \dot\cup I_{+-}, Q \subseteq \mathcal{P}(I),\\ K \in (\mathcal{P}(I_{+-} \dot\cup I^-)) - Q, K' \in (\mathcal{P}(I_{+-} \dot\cup I^-)) \cap Q|\!\}\end{array}} \qquad (32)$$

partition axiom for multisets *for* $X = X\langle\bar X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\}\rangle$ *and* $P \subseteq \mathcal{P}(I)$*:*

$$\frac{}{\begin{array}{l}\{\!|\{\lambda\} \to \{X\langle\bar X_{I'}\{\lambda\}\rangle \mid I' \cap I^+ \neq \emptyset\} \cup\\ \{X\langle\bar X_{J\cup J_-}\{\lambda\}\rangle, X\langle\bar X_J\{\lambda\}\rangle \mid J_- \subseteq I_-, J \in Q, J \subseteq I_{+-} \dot\cup I^-\},\\ \{\lambda\} \to \{X\langle\bar X_K\{\lambda\}\rangle\}, \{X\langle\bar X_{K'}\{\lambda\}\rangle\} \to \{X\} \mid\\ I = I^- \dot\cup I_+ \dot\cup I_- \dot\cup I_{+-}, Q \subseteq \mathcal{P}(I),\\ K \in (\mathcal{P}(I_{+-} \dot\cup I^-)) - Q, K' \in (\mathcal{P}(I_{+-} \dot\cup I^-)) \cap Q|\!\}\end{array}} \qquad (33)$$

Proof. The soundness proof for the weakening rule (24) is trivial.

For the left union rule (25) assume $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$, i.e. there exist $t_1, t_2 \in r$ such that for all $i \in I$ we get $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}_i$ and $\pi^X_{Z_i}(t_1) \neq \pi^X_{Z_i}(t_2)$ for some $Z_i \in \mathcal{Z}_i$. In particular, $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$, hence $r \not\models \{\!\{\mathcal{Y} \to \mathcal{Z}_i \mid i \in I\}\!\}$.

For the shift rule (26) assume $r \not\models \{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}$, i.e. there exist $t_1, t_2 \in r$ such that $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ and $\pi^X_Z(t_1) \neq \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$. Take a maximal $\mathcal{U}' \subseteq \mathcal{U}$ such that $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}'$. If we had $r \models \{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\}$, we would have $\mathcal{U}' \subsetneq \mathcal{U}$, and there would exist some $V \in \mathcal{U} - \mathcal{U}'$ with $\pi^X_V(t_1) = \pi^X_V(t_2)$, which contradicts the maximality of $\mathcal{U}$.

Let $X = X\{X_1(X_1') \oplus \cdots \oplus X_n(X_n')\} = X(X_1\{X_1'\}, \ldots, X_n\{X_n'\})$, $Y = X_I\{\lambda\}$, $Z_1 = X_J\{\lambda\}$ and $Z_2 = X(X(X_{i_1}\{X_{i_1}'\}, \ldots, X_{i_k}\{X_{i_k}'\})$ for the union axiom (27). Let $t_1, t_2 \in r$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ and $\pi^X_{Z_1}(t_1) \neq \pi^X_{Z_1}(t_2)$. Thus, one of t_1 or t_2 — without loss of generality let this be t_2 — must not contain elements of the form $(X_j : v_j)$ with $j \in J$. On the other hand, either t_1 and t_2 both contain elements of the form $(X_i : v_i)$ with $i \in I$ or both do not. As $I \subsetneq J$, it follows $\pi^X_{X(X_i\{\lambda\})}(t_1) = \pi^X_{X(X_i\{\lambda\})}(t_2) = \emptyset$ for all $i \in I$, which implies $\pi^X_{Z_2}(t_1) = \pi^X_{Z_2}(t_2)$.

For the partition axiom (28) let $t_1, t_2 \in r$ with $\pi^X_{X_I\{\lambda\}}(t_1) = \pi^X_{X_I\{\lambda\}}(t_2)$ and $\pi^X_{X(X_i\{\lambda\})}(t_1) \neq \pi^X_{X(X_i\{\lambda\})}(t_2)$ for all $i \in I$. Let $I_j \subseteq I$ be such that t_j contains an element of the form $(X_i : v_i)$ for all $i \in I_j$ $(j = 1, 2)$. Obviously, $I = I_1 \cup I_2$ and $\pi^X_{X_{I'}\{\lambda\}}(t_1) = \pi^X_{X_{I'}\{\lambda\}}(t_2)$ for all $I' \subseteq I$ with $I' \cap I_1 \neq \emptyset \neq I' \cap I_2$.

For the first plus/minus axiom in (29) let t_1, t_2 satisfy $\pi^X_{X_j\{\lambda\}}(t_1) = \pi^X_{X_j\{\lambda\}}(t_2)$ and $\pi^X_{X_k\{\lambda\}}(t_1) \neq \pi^X_{X_k\{\lambda\}}(t_2)$ for all $j \in I^+$ and $k \in I^-$. Assume that for all $i \in I^+$ there is some $J \subseteq I^-$ with $\pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{J\cup\{i\}}\{\lambda\}}(t_2)$, i.e. one of these projections must be $\emptyset$. As we have $\pi^X_{X_i\{\lambda\}}(t_1) = \pi^X_{X_i\{\lambda\}}(t_2)$, these must both be $\emptyset$, which implies $\pi^X_{X_{I^+}\{\lambda\}}(t_j) = \emptyset$ for $j = 1, 2$. Now $\pi^X_{X_k\{\lambda\}}(t_1) \neq \pi^X_{X_k\{\lambda\}}(t_2)$ for all $k \in I^-$, so if $\pi^X_{X_{I^-}\{\lambda\}}(t_1) \neq \pi^X_{X_{I^-}\{\lambda\}}(t_2)$ holds, one of these projections must be $\emptyset$ again, which implies that one t_j is $\emptyset$, the other not empty. That is $\pi^X_{X_{\{1,\ldots,n\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{\{1,\ldots,n\}}\{\lambda\}}(t_2)$.

For the second plus/minus axiom in (30) assume that it does not hold. Then we find two complex values t_1, t_2 that coincide on $X_{I\cup J}\{\lambda\}$, but differ on $X_J\{\lambda\}$ and all $X_{\{j\}}\{\lambda\}$ with $j \in J$. Furthermore, for each $i \in I$ there is at least one $J_i \subseteq J$ such that t_1, t_2 differ on $X_{J_i\cup\{i\}}\{\lambda\}$. It follows that one of the two complex values – without loss of generality let this be t_1 – contains values $(X_j : \tau_j)$ for all $j \in J$, while the other one does not contain such values. Then we obtain $\pi^X_{X_{J'\cup\{i\}}\{\lambda\}}(t_1) \neq \emptyset$ for all $J' \subseteq J$ and all $i \in I$. As t_1, t_2 coincide on $X_{I\cup J}\{\lambda\}$, this also gives $\pi^X_{X_{J'\cup\{i\}}\{\lambda\}}(t_2) \neq \emptyset$ for all $J' \subseteq J$ and at least one $i \in I$ contradicting the assumption that for at least one such $J' = J_i$ we have $\pi^X_{X_{J'\cup\{i\}}\{\lambda\}}(t_1) \neq \pi^X_{X_{J'\cup\{i\}}\{\lambda\}}(t_2)$.

For the third plus/minus axiom in (31) assume that it does not hold. Then we find two complex values t_1, t_2 that coincide on $X_{I^-}\{\lambda\}$, all $X_{\{i\}}\{\lambda\}$ for $i \in I^+$, and

on $X_{I' \cup J'}\{\lambda\}$, but differ on $X_{J'}\{\lambda\}$ and all $X_{\{k\}}\{\lambda\}$ with $k \in I^-$. Furthermore, for each $\ell \in I'$ there is at least one $J_\ell \subseteq I^-$ such that t_1, t_2 differ on $X_{J_\ell \cup \{\ell\}}\{\lambda\}$. Define $I_j^- = \{i \in I^- \mid \pi^X_{X_{\{i\}}\{\lambda\}}(t_j) \neq \emptyset\}$ $(j = 1, 2)$ to define a partition $I^- = I_1^- \cup I_2^-$. As t_1, t_2 differ on $X_{J'}\{\lambda\}$, this implies $J' \subseteq I_1'$ or $J' \subseteq I_2'$. Without loss of generality we can assume the first of these possibilities. As t_1, t_2 coincide on $X_{I' \cup J'}\{\lambda\}$, we must have $\pi^X_{X_{I'}\{\lambda\}}(t_2) \neq \emptyset$, so also $\pi^X_{X_{\{i\}}\{\lambda\}}(t_2) \neq \emptyset$ for some $i \in I'$. Then also $\pi^X_{X_{\{i\}}\{\lambda\}}(t_1) \neq \emptyset$ due to $I' \subseteq I^+$. Hence we get $\pi^X_{X_{J \cup \{i\}}\{\lambda\}}(t_j) \neq \emptyset$ for $j = 1, 2$ and all $J \subseteq I^-$ contradicting the assumption that at least one such $J = J_i$ exists, such that t_1, t_2 differ on $X_{J_i \cup \{i\}}\{\lambda\}$.

For the set partition axiom in (32) take any $S_1, S_2 \in dom(X)$. In case $S_1 = S_2 = \emptyset$ we simple choose $I_+ = I$, so we must have $I^- = I_- = I_{+-} = \emptyset$. Further take $Q' = \{\emptyset\}$. In case exactly one of the S_i is empty, we choose $I^- = I$, $I_+ = I_- = I_{+-} = \emptyset$, and

$$Q' = \{J \subseteq I^- \mid \pi^X_{X\{\bar{X}_J\{\lambda\}\}}(S_1) = \pi^X_{X\{\bar{X}_J\{\lambda\}\}}(S_2)\}.$$

In both cases we immediately get the satisfaction of the first involved FD, if $Q \cap (\mathcal{P}(I_{+-} \cup I^-)) = Q'$. However, if there is some $K \in Q'$ with $K \notin Q$, the FD $\{\lambda\} \to \{X\{\bar{X}_K\{\lambda\}\}\}$ is satisfied. Similarly, if there is some $K' \in Q$ with $K' \notin Q'$, then the FD $\{X\{\bar{X}_{K'}\{\lambda\}\}\} \to \{X\}$ is satisfied.

In the remaining case with $S_1 \neq \emptyset \neq S_2$ we take

$$\begin{aligned}
I_+ &= \{i \in I \mid \pi^X_{X\{\bar{X}_{\{i\}}\{\lambda\}\}}(S_1) = \{\{\top\}\} = \pi^X_{X\{X_{\{i\}}\{\lambda\}\}}(S_2)\},\\
I_- &= \{i \in I \mid \pi^X_{X\{\bar{X}_{\{i\}}\{\lambda\}\}}(S_1) = \{\emptyset\} = \pi^X_{X\{X_{\{i\}}\{\lambda\}\}}(S_2)\},\\
I^- &= \{i \in I \mid \pi^X_{X\{\bar{X}_{\{i\}}\{\lambda\}\}}(S_1) \neq \pi^X_{X\{X_{\{i\}}\{\lambda\}\}}(S_2)\},
\end{aligned}$$

and $I_{+-} = I - I^- - I_+ - I_-$. Then S_1, S_2 obviously coincide on all $X\{\bar{X}_{I'}\{\lambda\}\}$ with $I' \cap I^+ \neq \emptyset$. If we take again

$$Q' = \{J \subseteq I^- \cup I_{+-} \mid \pi^X_{X\{\bar{X}_J\{\lambda\}\}}(S_1) = \pi^X_{X\{\bar{X}_J\{\lambda\}\}}(S_2)\},$$

then S_1, S_2 coincide on all $X\{\bar{X}_{J \cup J_-}\{\lambda\}\}$ and all $X\{\bar{X}_J\{\lambda\}\}$ with $J_- \subseteq I_-$ and $J \in Q'$. As in the previous two cases we obtain the satisfaction of the first involved FD, if $Q \cap (\mathcal{P}(I_{+-} \cup I^-)) = Q'$ holds. If this is not the case, one of the other FDs will be satisfied by $\{S_1, S_2\}$.

Finally, for the multiset partition axiom in (33) we proceed analogously. Let $M_1, M_2 \in dom(X)$. In case $M_1 = M_2 = \langle\rangle$ we simple choose $I_+ = I$, so we must have $I^- = I_- = I_{+-} = \emptyset$. Further take $Q' = \{\emptyset\}$. In case exactly one of the M_i is the empty multiset, we choose $I^- = I$, $I_+ = I_- = I_{+-} = \emptyset$, and $Q' = \{J \subseteq I^- \mid \pi^X_{X\langle\bar{X}_J\{\lambda\}\rangle}(M_1) = \pi^X_{X\langle\bar{X}_J\{\lambda\}\rangle}(M_2)\}$.

In case $M_1 \neq \langle\rangle \neq M_2$ we take

$$I_+ = \{i \in I \mid \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_1) = \langle \underbrace{\{\top\}}_{x \text{ times}} \rangle = \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_2)\},$$

$$I_- = \{i \in I \mid \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_1) = \langle \underbrace{\emptyset}_{x \text{ times}} \rangle = \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_2)\},$$

$$I^- = \{i \in I \mid \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_1) \neq \pi^X_{X\langle \bar{X}_{\{i\}}\{\lambda\}\rangle}(M_2)\},$$

and $I_{+-} = I - I^- - I_+ - I_-$. As before we define $Q' = \{J \subseteq I^- \cup I_{+-} \mid \pi^X_{X\langle \bar{X}_J\{\lambda\}\rangle}(M_1) = \pi^X_{X\langle \bar{X}_J\{\lambda\}\rangle}(M_2)\}$.

In all three cases M_1, M_2 coincide on all $X\langle \bar{X}_{I'}\{\lambda\}\rangle$ with $I' \cap I^+ \neq \emptyset$, on all $X\langle \bar{X}_{J \cup J_-}\{\lambda\}\rangle$ and all $X\langle \bar{X}_J\{\lambda\}\rangle$ with $J_- \subseteq I_-$ and $J \in Q'$. Hence $\{M_1, M_2\}$ satisfies the first involved FD, if $Q \cap (\mathcal{P}(I_{+-} \cup I^-)) = Q'$ holds, while for other Q one of the other FDs will be satisfied.

□

Note that the first three rules (24), (25) and (26) in Theorem 6 are a slight generalisation of the rules used for wFDs in the RDM (see e.g. [32, p.100f.]). The other axioms (27) – (33) arise again from the structure of the subattribute lattice.

4.2 The Completeness Theorem for the Derivation of wFDs

We now want to show that the axioms and rules for the implication of wFDs in Theorems 3, 5 and 6 are also complete. This gives our main result. Before we come to the proof let us make a little observation on the union-constructor.

If $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$, then each instance r of X can be partioned into r_i $(i = 1, \ldots, n)$, where r_i contains exactly the X_i-labelled elements of r. Then r satisfies a FD $\varphi \equiv \mathcal{Y} \to \mathcal{Z}$ iff each r_i satifies the i'th projection φ_i of φ, which results by replacing all subattributes $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ in $\mathcal{Y}$ or $\mathcal{Z}$ by $X_i(Y_i)$. Similarly, we see $\varphi \in \Sigma^+$ iff $\varphi_i \in \Sigma_i^+$ for all $i = 1, \ldots, n$.

Lemma 1. *Let $r \subseteq dom(X)$ be an instance of $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$ and let $\mathcal{Y} \to \mathcal{Z}$ be a FD on $\mathcal{S}(X)$. Define $r_i = \{(X_i : v_i) \mid (X_i : v_i) \in r\}$, $\mathcal{Y}_i = \{X_i(Y_i) \mid X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \in \mathcal{Y}$ for some $Y_j (j = 1, \ldots, n, j \neq i)\}$, and $\mathcal{Z}_i = \{X_i(Z_i) \mid X_1(Z_1) \oplus \cdots \oplus X_n(Z_n) \in \mathcal{Z}$ for some $Z_j (j = 1, \ldots, n, j \neq i)\}$ $(i = 1, \ldots, n)$. Furthermore, for a set Σ of FDs on $\mathcal{S}(X)$ let $\Sigma_i = \{\mathcal{Y}_i \to \mathcal{Z}_i \mid \mathcal{Y} \to \mathcal{Z} \in \Sigma\}$. Then the following holds:*

1. *$r \models \mathcal{Y} \to \mathcal{Z}$ iff $r_i \models \mathcal{Y}_i \to \mathcal{Z}_i$ holds for all $i = 1, \ldots, n$;*
2. *$\mathcal{Y} \to \mathcal{Z} \in \Sigma^+$ iff $\mathcal{Y}_i \to \mathcal{Z}_i \in \Sigma_i^+$ for all $i = 1, \ldots, n$.*

Proof. For the first claim let us first assume $r \models \mathcal{Y} \to \mathcal{Z}$. Take $t_1 = (X_i : t_1') \in r_i$ and $t_2 = (X_i : t_2') \in r_i$ with $\pi^{X_i'}_{Y_i}(t_1') = \pi^{X_i'}_{Y_i}(t_2')$ for all Y_i with $X_i(Y_i) \in \mathcal{Y}_i$. Then $\pi^X_{X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)}(t_1) = \pi^X_{X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)}(t_2)$ holds for all $X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \in$

$\mathcal{Y}$, and thus $r \models \mathcal{Y} \to \mathcal{Z}$ implies $\pi^X_{X_1(Z_1)\oplus\cdots\oplus X_n(Z_n)}(t_1) = \pi^X_{X_1(Z_1)\oplus\cdots\oplus X_n(Z_n)}(t_2)$ for all $X_1(Z_1) \oplus \cdots \oplus X_n(Z_n) \in \mathcal{Z}$. This gives $\pi^{X'_i}_{Z_i}(t'_1) = \pi^{X'_i}_{Z_i}(t'_2)$ for all Z_i with $X_i(Z_i) \in \mathcal{Z}_i$, hence $r_i \models \mathcal{Y}_i \to \mathcal{Z}_i$ holds for all $i = 1, \ldots, n$.

Conversely, assume $r_i \models \mathcal{Y}_i \to \mathcal{Z}_i$ holds for all $i = 1, \ldots, n$, and take $t_1 = (X_i : t'_1) \in r$ and $t_2 = (X_j : t'_2) \in r$. If $i \neq j$, then t_1, t_2 differ on all subattributes except λ, i.e. $\{t_1, t_2\} \models \mathcal{Y} \to \mathcal{Z}$. So assume $j = i$, i.e. $t_1, t_2 \in r_i$, and $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$, i.e. for $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n)$ we obtain $\pi^{X'_i}_{Y_i}(t'_1) = \pi^{X'_i}_{Y_i}(t'_2)$. As $X_i(Y_i) \in \mathcal{Y}_i$, the premise implies $\pi^{X'_i}_{Z_i}(t'_1) = \pi^{X'_i}_{Z_i}(t'_2)$ for all Z_i with $X_i(Z_i) \in \mathcal{Z}_i$ and further $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$, hence $r \models \mathcal{Y} \to \mathcal{Z}$.

For the second claim first assume $\mathcal{Y}_i \to \mathcal{Z}_i \in \Sigma_i^+$ for all $i = 1, \ldots, n$. Then $\mathcal{Y} \to \mathcal{Z} \in \Sigma^+$ results from successive applications of the union lifting rule (21) together with the subattribute axiom (2), the reflexivity axiom (1) and the transitivity rule (6).

Conversely, in a derivation of $\mathcal{Y} \to \mathcal{Z}$ from Σ all involved subattributes other than λ will have the form $X_1(U_1) \oplus \cdots \oplus X_n(U_n)$ with $X'_i \geq U_i$. Reducing this to $X_i(U_i)$ in each step gives a valid derivation of $\mathcal{Y}_i \to \mathcal{Z}_i$ from Σ_i.

□

Theorem 7 (Completeness Theorem). *The set of axioms and rules in Theorems 3, 5 and 6 is complete for the implication of wFDs on* $\mathcal{S}(X)$.

Proof. Let Σ be a set of wFDs on $\mathcal{S}(X)$ and assume $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^+$. Due to the union rule (7) we must have $\{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ for some selected $Z_i \in \mathcal{Z}_i$. Furthermore, due to the left union rule (25) we get $\{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ with $\mathcal{Y} = \bigcup_{i \in I} \mathcal{Y}_i$.

Let $\mathcal{Z} = \{Z \mid Z \geq Z_i \text{ for some } i \in I\}$ and $\mathcal{U} = \mathcal{S}(X) - \mathcal{Y} - \mathcal{Z}$. Due to the reflexivity axiom (1) we obviously have $Z_i \notin \mathcal{Y}$, and then $\mathcal{Y} \sqcap \mathcal{Z} = \emptyset$ due to the subattribute axiom (2). Due to the shift rule (26) there must exist some $\mathcal{U}' \subseteq \mathcal{U}$ with $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$. Otherwise we could derive $\{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}$, and thus $\{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \in \Sigma^+$ contradicting our assumption.

Lemma 2. *Let $\mathcal{U}'$ be maximal with the given property. Then $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$ is a coincidence ideal.*

We first prove Lemma 2, then continue the proof of Theorem 7.

of Lemma 2.

1. Assume $\mathcal{F} = \emptyset$. This implies $\mathcal{Z} \cup \mathcal{U} = \mathcal{S}(X)$ and thus $\{\!\{\emptyset \to \{Z\} \mid Z \in \mathcal{S}(X)\}\!\} \notin \Sigma^+$. This wFD, however, can be derived from $\emptyset \to \{\lambda\} \in \Sigma^+$ (due to the λ-axiom (4)) using the weakening rule (24). Thus, $\mathcal{F}$ is not empty.

2. Now let $Y \in \mathcal{F}$ and $Y \geq Y'$ Assume $Y' \notin \mathcal{F}$. So $Y' \in \mathcal{U}$, otherwise we get $Y' \in \mathcal{Y} \cup \mathcal{Z}$, which implies $Y' \geq Z_i$ for some $i \in I$ and furtheron $Y \geq Y' \geq Z_i$, which gives the contradiction $Y \in \mathcal{Z}$.

Now take $\mathcal{U}'' = \mathcal{U}' \cup \{Y'\}$. The subattribute axiom (2) together with the extension and transitivity rules (5) and (6) implies $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{Y} \cup \mathcal{U}'' \in \Sigma^+$. As $\mathcal{U}'$ was chosen maximal, we also have $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$. Using the transitivity rule (6) again, this gives $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$. Then the weakening rule (24) leads to the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.

3. Let $Y_1, Y_2 \in \mathcal{F}$ be reconsilable. Assume $Y = Y_1 \sqcup Y_2 \notin \mathcal{F}$. If $Y \in \mathcal{U}$, we take $\mathcal{U}'' = \mathcal{U}' \cup \{Y\}$. Due to the maximality of $\mathcal{U}'$ we get $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'')\}\!\} \in \Sigma^+$, thus by the weakening rule (24) also $\{\!\{\mathcal{Y} \cup \mathcal{U}'' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.

   On the other hand, the join axiom (3) implies $\{Y_1, Y_2\} \to \{Y\} \in \Sigma^+$. Using the reflexivity axiom (1), the extension rule (5) and the transitivity rule (6) we obtain $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{Y} \cup \mathcal{U}'' \in \Sigma^+$, from which we get the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$ by another application of the transitivity rule.

   If $Y \notin \mathcal{U}$, we get $Y \in \mathcal{Z}$, thus $\{Y\}$ is among the right hand sides in $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$. However, the join rule (9) together with the reflexivity axiom and the transitivity rule imply $\mathcal{Y} \cup \mathcal{U}' \to \{Y\} \in \Sigma^+$, hence the weakening rule leads to the contradiction $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$.

4. a) Assume $X_I\{\lambda\} \in \mathcal{F}$, but $(X_J\{\lambda\} \notin \mathcal{F}$ for $\{i_1, \ldots, i_k\} = I \subsetneq J$. As $\mathcal{S}(X)$ is partitioned into $\mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$ and $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$, we must have $X_J\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. From the union axiom (27), the transitivity rule and $X(X_I\{\lambda\}) \in \mathcal{F}$ we conclude

$$\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \{X_J\{\lambda\}, X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\})\}\}\!\} \in \Sigma^+.$$

   Due to the weakening rule (24) it follows $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{W}\}\!\} \in \Sigma^+$ for all $\mathcal{W} \subseteq \mathcal{S}(X)$ with $X_J\{\lambda\}, X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \in \mathcal{W}$. According to the definition of $\mathcal{U}'$ we must have either $X_J\{\lambda\} \notin \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$ or $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \notin \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, which implies $X(X_{i_1}\{X'_{i_1}\}, \ldots, X_{i_k}\{X'_{i_k}\}) \in \mathcal{F}$.

   b) Assume $X_I\{\lambda\} \in \mathcal{F}$, but $X(X_i\{\lambda\}) \notin \mathcal{F}$ for all $i \in I$. In particular $X(X_i\{\lambda\}) \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. Using the partition axiom (28), the transitivity rule and $X_I\{\lambda\} \in \mathcal{F}$ we conclude $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{X_{I'_1 \cup I'_2}\{\lambda\} \mid \emptyset \neq I'_1 \subseteq I_1, \emptyset \neq I'_2 \subseteq I_2\}, \mathcal{Y} \cup \mathcal{U}' \to \{X(X_i\{\lambda\})\} \mid I = I_1 \dot\cup I_2, i \in I, I_1 \neq \emptyset \neq I_2\}\!\} \in \Sigma^+$.

   If for all partitions $I = I_1 \dot\cup I_2$ we had at least one $X_{I'_1 \cup I'_2}\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, we can apply the reflexivity axiom, the transitivity rule and the weakening rule to derive $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \in \Sigma^+$ contradicting the assumption on $\mathcal{U}'$. Therefore, there is a partition $I = I_1 \dot\cup I_2$ with $\{X_{I'_1 \cup I'_2}\{\lambda\} \mid \emptyset \neq I'_1 \subseteq I_1, \emptyset \neq I'_2 \subseteq I_2\} \subseteq \mathcal{F}$.

Choose such a partition. If we had $X_{I_1}\{\lambda\} \in \mathcal{F}$, we could choose a maximal $J \subsetneq I_1$ with $X_J\{\lambda\} \notin \mathcal{F}$, and $X_{\{j\}}\{\lambda\} \notin \mathcal{F}$ for all $j \in J$. So, we can partition I_1 into J and $I' = I_1 - J$. Now use property 4(d) – which we prove soon not using 4(b). Due to this property we find some $i \in I'$ such that $X_{J' \cup \{i\}}\{\lambda\} \in \mathcal{F}$ holds for all $J' \subseteq J$. In particular, for $J' = \emptyset$ we obtain a contradiction. Hence we must have $X_{I_1}\{\lambda\} \notin \mathcal{F}$ and by symmetry also $X_{I_2}\{\lambda\} \notin \mathcal{F}$.

c) Assume $X_{\{1,\ldots,n\}}\{\lambda\} \in \mathcal{F}$, $X_{I^-}\{\lambda\} \notin \mathcal{F}$ and for all $i \in I^+$ there is some $J \subseteq I^-$ with $X_{J \cup \{i\}}\{\lambda\} \notin \mathcal{F}$. Let this J be denoted as J_i. Taking the first plus/minus axiom (29), the left hand side of the FDs are always in $\mathcal{F}$. Therefore, using the reflexivity axiom and the transitivity rule we derive $\{\!\{\mathcal{F} \to \{X_{J_i \cup \{i\}}\{\lambda\}\}, \mathcal{F} \to \{X\}, \mathcal{F} \to \{X_j\{\lambda\}\}, \mathcal{F} \to \{X_{I^-}\{\lambda\}\} \mid i \in I^+, j \in I^-\!\} \in \Sigma^+$. Now the right hand sides of the FDs are all not in $\mathcal{F}$, so the weakening rule implies $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\!\} \in \Sigma^+$ contradicting the construction of $\mathcal{F}$, according to which $\mathcal{S}(X) - \mathcal{F} = \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. and $\{\!\{\mathcal{F} \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\!\} \notin \Sigma^+$.

d) Let $I \cap J = \emptyset$ and $X_J\{\lambda\} \notin \mathcal{F}$, $X_{\{j\}}\{\lambda\} \notin \mathcal{F}$ for all $j \in J$, and for all $i \in I$ there is some $J_i \subseteq J$ with $X_{J_i \cup \{i\}}\{\lambda\} \notin \mathcal{F}$. Furthermore, assume $X_{I \cup J}\{\lambda\} \in \mathcal{F}$. Then from the second plus/minus axiom (30), the transitivity rule (6) and the reflexivity axiom (1) we derive $\{\!\{\mathcal{F} \to \{X_J\{\lambda\}\}, \mathcal{F} \to \{X_{\{j\}}\{\lambda\}\}, \mathcal{F} \to \{X_{J_i \cup \{i\}}\{\lambda\}\} \mid i \in I, j \in J\!\} \in \Sigma^+$. Here the right hand sides of all involved FDs have the form $\{Z\}$ with $Z \notin \mathcal{F}$, so the weakening rule (24) gives $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\!\} \in \Sigma^+$ contradicting the construction of $\mathcal{F}$. Hence we must have $X_{I \cup J}\{\lambda\} \notin \mathcal{F}$.

e) Assume $X_{I^-}\{\lambda\} \in \mathcal{F}$, and let $I' \subseteq I^+$ such that for all $i \in I'$ there is some $J_i \subseteq I^-$ with $X_{J_i \cup \{i\}}\{\lambda\} \notin \mathcal{F}$. Let $J' \subseteq I^-$ with $X_{J'}\{\lambda\} \notin \mathcal{F}$ and assume $X_{I' \cup J'}\{\lambda\} \in \mathcal{F}$. Then for $i \in I'$, $k \in I^-$ using the the third plus/minus axiom (31), the reflexivity axiom (1) and the transitivity rule (6) we derive $\{\!\{\mathcal{F} \to \{X_{J'}\{\lambda\}\}, \mathcal{F} \to \{X_{J_i \cup \{i\}}\{\lambda\}\}, \mathcal{F} \to \{X_{\{k\}}\{\lambda\}\}\!\} \in \Sigma^+$. Again the right hand sides of all involved FDs have the form $\{Z\}$ with $Z \notin \mathcal{F}$ leading to the contradiction $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\!\} \in \Sigma^+$ by applying the weakening rule (24). Hence we must have $X_{I' \cup J'}\{\lambda\} \notin \mathcal{F}$ for all $J' \subseteq I^-$ with $X_{J'}\{\lambda\} \notin \mathcal{F}$.

5. a) Let $X_I\{\lambda\}, X_J\{\lambda\} \in \mathcal{F}$ with $I \cap J = \emptyset$, but assume $X_{I \cup J}\{\lambda\} \notin \mathcal{F}$, i.e. $X_{I \cup J}\{\lambda\} \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$. From the set axiom (10), the reflexivity axiom (1) and the transitivity rule (6) we derive $\mathcal{F} \to \{X_{I \cup J}\{\lambda\}\} \in \Sigma^+$ and further $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\!\} \in \Sigma^+$ by the weakening rule (24). This contradicts the construction of $\mathcal{F}$, so we must have $X_{I \cup J}\{\lambda\} \in \mathcal{F}$.

The proof of properties 5(b) and (c) is completely analogous using (15) and (11), respectively, instead of (10).

d) Let $X_I[\lambda], X_J[\lambda] \in \mathcal{F}$ with $I \subseteq J$, but assume $X_{J-I}[\lambda] \notin \mathcal{F}$. From the second list axiom (16), the reflexivity axiom (1) and the transitivity rule (6) we derive $\mathcal{F} \to \{X_{J-I}[\lambda]\} \in \Sigma^+$. Applying the weakening rule (24)

leads to the contradiction $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$. Hence we must have $X_{J-I}[\lambda] \in \mathcal{F}$.

The proof of property 5(e) is completely analogous using (12) instead of (16).

f) Let $X_I[\lambda], X_J[\lambda] \in \mathcal{F}$ and assume $X_{I \cap J}[\lambda] \in \mathcal{F}$, but $X_{(I \cup J)-(I \cap J)}[\lambda] \notin \mathcal{F}$. The the third list axiom (17), the reflexivity axiom (1) and the transitivity rule (6) allow us to derive $\mathcal{F} \to \{X_{(I \cup J)-(I \cap J)}[\lambda]\} \in \Sigma^+$. Further application of the weakening rule (24) leads to the contradiction $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$. Hence we must have $X_{(I \cup J)-(I \cap J)}[\lambda] \in \mathcal{F}$.

Analogously, assuming $X_{I \cap J}[\lambda] \notin \mathcal{F}$ and $X_{(I \cup J)-(I \cap J)}[\lambda] \in \mathcal{F}$ leads to the same contradiction using the fourth list axiom (18) instead of (17). The proof of property 5(g) is completely analogous using (13) and (14) instead of (17) and (18), respectively.

6. If property 6(a) were not satisfied, then for all partitions $I \cap I^+ = I_+ \cup I_- \cup I_{+-}$ one of the properties ii or iii in Definition 7 6(a) would be violated. In case property ii is violated there is some I' with $I' \cap I_+ \neq \emptyset$ and $X\{\bar{X}_{I'}\{\lambda\}\} \notin \mathcal{F}$. In case property iii is violated there exists some $I' \subseteq I_{+-} \cup I^- \cup I_-$ such that either $X\{\bar{X}_{I'}\{\lambda\}\} \in \mathcal{F}$ and $X\{\bar{X}_{I' \cap (I_{+-} \cup I^-)}\{\lambda\}\} \notin \mathcal{F}$ or $X\{\bar{X}_{I'}\{\lambda\}\} \notin \mathcal{F}$ and $X\{\bar{X}_{I' \cap (I_{+-} \cup I^-)}\{\lambda\}\} \in \mathcal{F}$. Then define $J_- = I' - I_{+-} - I^-$ and $J = I' - I_-$, which gives $X\{\bar{X}_J\{\lambda\}\} \notin \mathcal{F}$ in the first case and $X\{\bar{X}_{J_- \cup J}\{\lambda\}\} \notin \mathcal{F}$ in the second case.

Let $Q = \{J \subseteq I \mid X\{\bar{X}_J\{\lambda\}\} \in \mathcal{F}\}$. Then the right hand side of the first FD in the set partition axiom (32) contains a subattribute $Z \notin \mathcal{F}$, and the same holds for the involved FDs of the form $\{\lambda\} \to \{X\{\bar{X}_K\{\lambda\}\}\}$. For the remaining involved FDs we can replace the right hand side by $\{Z\}$ with some $Z \notin \mathcal{F}$ using the subattribute axiom (2) and the transitivity rule. Thus, using (32), (2), the reflexivity axiom, the transitivity rule, and the weakening rule, we derive the contradiction $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$. Hence there is a partition $I = I^- \cup I_+ \cup I_- \cup I_{+-}$ satisfying the properties i, ii or iii in Definition 7 6(a).

The proof of property 6(b) is completely analogous using the multiset partition axiom (33) instead of (32).

7. For the proof of property 7 observe that the proofs of properties 1 - 6 follow a simple pattern. Assuming that the property does not hold we obtain an instance of a particular axiom, which together with the reflexivity axiom (1), the transitivity rule (6) and the weakening rule (24) allows us to derive the contradiction $\{\!\{\mathcal{F} \to \{Z\} \mid Z \notin \mathcal{F}\}\!\} \in \Sigma^+$.

To be precise, we used the λ axiom (4) for property 1, the subattribute axiom (2) for property 2, the join axiom (3) for property 3, the union axiom (27) for property 4(a), the partition axiom (28) for property 4(b), the first plus/minus axiom (29) for property 4(c), the second plus/minus axiom (30) for property 4(d), the third plus/minus axiom (31) for property 4(e), the set axiom (10)

for property 5(a), the four list axioms (15) – (18) for properties 5(b),(d) and (f), the four multiset axioms (11) – (14) for properties 5(c),(e) and (g), the set partition axiom (32) for property 6(a), and the multiset partition axiom (33) for property 6(b).

We can apply the record lifting rule (20), the union lifting rule (21) and the list lifting rule (23) to all these axioms to derive additional axioms, and we can apply the set lifting rule (19) and the multiset lifting rule (22) to the axioms except (3), (31) and (10) – (18). The resulting axioms differ from the original ones only by "wrapping" constructors around the involved attributes. Then using exactly the same arguments as before, we obtain additional properties for $\mathcal{F}$ that correspond to the required properties for the embedded ideals $\mathcal{F}_i$ or $\mathcal{G}$ used in properties 7(a)-(e) and 8(a)-(c), which completes the proof.

□

Proof of Theorem 7 (continued): Due to the restructuring rules in Definition 4 we may assume that the union-constructor appears in X only inside a set-, list- or multiset-constructor or as the outermost constructor.

Let us first assume that the outermost constructor is not the union-constructor. Then we can apply the Central Theorem 2, which gives us $r = \{t_1, t_2\} \subseteq dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y \in \mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$. In particular, $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $i \in I$ and $Y \in \mathcal{Y}_i$, and $\pi^X_{Z_i}(t_1) \neq \pi^X_{Z_i}(t_2)$ for all $i \in I$. That is, $r \not\models \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\}$. From the soundness of the fragmentation rule (8) we conclude $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

Now assume that the outermost constructor of X is the union-constructor, say $X = X_1(X'_1) \oplus \cdots \oplus X_n(X'_n)$. We know from Lemma 2 that $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$ is a coincidence ideal on $\mathcal{S}(X)$. If $\mathcal{F} = \{\lambda\}$, then take $t_1 = (X_1 : t'_1)$ and $t_2 = (X_2 : t'_2)$ with arbitrary $t'_j \in dom(X'_j)$. Then $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y = \lambda$. As before this implies $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with $r = \{t_1, t_2\}$.

For $\mathcal{F} \neq \{\lambda\}$ take the embedded coincidence ideal $\mathcal{F}_i$ on $\mathcal{S}(X'_i)$ according to Definition 7. Using the Central Theorem 2 we find $t_{i1}, t_{i2} \in dom(X'_i)$ with $\pi^{X'_i}_{Y_i}(t_{i1}) = \pi^{X'_i}_{Y_i}(t_{i2})$ iff $Y_i \in \mathcal{F}_i$.

As we have $\{\!\{\mathcal{F} \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$, we must also have $\{\!\{\mathcal{F}_j \to \{Z\} \mid Z \in (\mathcal{Z} \cup (\mathcal{U} - \mathcal{U}'))_j\}\!\} \notin \Sigma^+_j$ for at least one j according to Lemma 1. In particular, for $Z_i = X_1(Z'_{i1}) \oplus \cdots \oplus X_n(Z'_{in})$ we find some j such that $Z'_{ij} \notin \mathcal{F}_j$ for all $i \in I$.

Now take $r = \{(X_j : t_{j1}), (X_j : t_{j2})\}$. Then for all $i \in I$ and all $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \in \mathcal{Y}_i \subseteq \mathcal{F}$ we have $Y_j \in \mathcal{F}_j$, and we obtain

$$\pi^X_Y((X_j : t_{j1})) = (X_j : \pi^{X'_j}_{Y_j}(t_{j1})) = (X_j : \pi^{X'_j}_{Y_j}(t_{j2})) = \pi^X_Y((X_j : t_{j2}))\ .$$

On the other hand, $Z'_{ij} \notin \mathcal{F}_j$ implies

$$\pi^X_{Z_i}((X_j : t_{j1})) = (X_j : \pi^{X'_j}_{Z'_{ij}}(t_{j1})) \neq (X_j : \pi^{X'_j}_{Z'_{ij}}(t_{j2})) = \pi^X_{Z_i}((X_j : t_{j2}))$$

for all $i \in I$. That is $r \not\models \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\}$, and hence $r \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ by the soundness of the fragmentation rule (8).

The next Lemma 3 shows $r \models \Sigma$ in both cases. This implies $r \models \Sigma^*$, and thus $\{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^*$, which completes the proof of Theorem 7.

□

Lemma 3. $r \models \Sigma$.

Proof. First assume again that the outermost constructor is not the union-constructor. Let $\{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\} \in \Sigma$.

1. If $\mathcal{V}_j \not\subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$, we get $\pi_V^X(t_1) \neq \pi_V^X(t_2)$ for some $V \in \mathcal{V}_j$. Thus $r \models \mathcal{V}_j \to \mathcal{W}_j$ and due to the soundness of the weakening rule also $r \models \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$.

2. If $\mathcal{V}_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for all $j \in J$, we get $\mathcal{Y} \cup \mathcal{U}' \to \mathcal{V}_j \in \Sigma^+$ from the reflexivity axiom, $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \mathcal{W}_j \mid j \in J\}\!\} \in \Sigma^+$ from the transitivity rule, and $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{W_j\} \mid j \in J\}\!\} \in \Sigma^+$ for any choices $W_j \in \mathcal{W}_j$ from the fragmentation rule.

   Assume we could select $W_j \in \mathcal{W}_j - \mathcal{Y} - \mathcal{U}'$ for all $j \in J$. Then the weakening rule implies $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{W\} \mid W \in \mathcal{S}(X) - \mathcal{Y} - \mathcal{U}'\}\!\} \in \Sigma^+$. However, $\mathcal{S}(X) - \mathcal{Y} - \mathcal{U}' = \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')$, so we get a contradiction to the choice of $\mathcal{U}'$.

   Therefore, we must have $\mathcal{W}_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$. By construction of r we get $\pi_W^X(t_1) = \pi_W^X(t_2)$ for all $W \in \mathcal{W}_j$, thus $r \models \mathcal{V}_j \to \mathcal{W}_j$. This implies $r \models \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$ due to the soundness of the weakening rule.

If the outermost constructor is the union-constructor, then according to Lemma 1 we have to show $r_j \models \Sigma_j$. The proof is analogous to the case before.

□

4.3 The Case of Functional Dependencies

Theorem 7 shows the axiomatisation of wFDs. If Σ is a set of "ordinary" FDs, we can apply the axioms and rules to Σ and then the FDs in Σ^+ will be the implied FDs. Of course, we would like to have an axiomatisation for FDs that avoids such a detour via the wFDs.

We first observe that most of the axioms and rules for wFDs in Theorem 6 depend on the joint occurrence of the set and the union constructor. Only the weakening rule (24), the left union rule (25) and the shift rule (26) do not make such a special assumption. In particular, in a derivation of FDs for a nested attribute X that does not contain both the union and the set constructor, the special axioms in Theorem 6 will not be needed. We now show that indeed none of the rules for the derivation of wFDs are needed either, i.e. the set of axioms and rules in Theorems 3 and 5 excluding the set axiom (10) are sound and complete for the derivation of FDs in this case.

In order to prove this, observe that properties 4, 5(a) and 6 in Theorem 2 can be ignored, if the union and the set constructor do not appear jointly.

Theorem 8. *Let $X \in \mathcal{N}$ be a nested attribute not containing both the set and the union constructor. Then the set of axioms and rules in Theorems 3, 5 excluding the set axiom (10) is complete for the implication of FDs on $\mathcal{S}(X)$.*

Proof. Let Σ be a set of FDs on $\mathcal{S}(X)$ and assume $\mathcal{Y} \to \mathcal{Z} \notin \Sigma^+$. Then due to the union rule (7) there exists a subattribute $Z \in \mathcal{Z}$ with $\mathcal{Y} \to \{Z\} \notin \Sigma^+$. Thus, $Z \notin \bar{\mathcal{Y}} = \{Z' \mid \mathcal{Y} \to \{Z'\} \in \Sigma^+\}$. We show that $\mathcal{F} = \bar{\mathcal{Y}}$ is a coincidence ideal on $\mathcal{S}(X)$:

1. $\lambda \in \mathcal{F}$ follows immediately from the reflexivity axiom (1), the λ axiom (4), and the transitivity rule (6).

2. For $Z_1 \in \mathcal{F}$ and $Z_1 \geq Z_2$ the subattribute axiom (2) and the transitivity rule (6) imply $Z_2 \in \mathcal{F}$.

3. For reconsilable $Z_1, Z_2 \in \mathcal{F}$ the join axiom (3) and the transitivity rule (6) imply $Z_1 \sqcup Z_2 \in \mathcal{F}$.

5. Property (b)–(g) result immediately from applying the multiset axioms (11) – (14) and the list axioms (15) – (18) together with the transitivity rule (6).

7. The proof of property 7 in Definition 7 is analogous to the corresponding proof for Lemma 2. We apply lifting rules (19) – (23) to the axioms used in the proof of properties 1, 2, 3 and 5, then apply the same argument as before.

If the outermost constructor is not the union-constructor, we can apply the Central Theorem 2, which gives us $r = \{t_1, t_2\} \subseteq dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y \in \mathcal{F}$. In particular, $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$, and $\pi^X_Z(t_1) \neq \pi^X_Z(t_2)$. That is, $r \not\models \mathcal{Y} \to \{Z\}$. From the soundness of the fragmentation rule (8) we conclude $r \not\models \mathcal{Y} \to \mathcal{Z}$.

If the outermost constructor of X is the union-constructor, say $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$ with $n \geq 2$, then either $\mathcal{F} = \{\lambda\}$ or we obtain embedded coincidence ideals $\mathcal{F}_i$ on $\mathcal{S}(X_i')$ $(i = 1, \ldots, n)$ according to Definition 7: In the first case take $t_1 = (X_1 : t_1')$ and $t_2 = (X_2 : t_2')$ with arbitrary $t_j' \in dom(X_j')$. Then $\pi^X_U(t_1) = \pi^X_U(t_2)$ iff $U = \lambda$. As before $Z \notin \mathcal{F}$ implies $r \not\models \mathcal{Y} \to \mathcal{Z}$ with $r = \{t_1, t_2\}$.

In the second case the Central Theorem 2 gives us $t_{i1}, t_{i2} \in dom(X_i')$ with $\pi^{X_i'}_{Y_i}(t_{i1}) = \pi^{X_i'}_{Y_i}(t_{i2})$ iff $Y_i \in \mathcal{F}_i$. As we have $\mathcal{F} \to \{Z\} \notin \Sigma^+$, we must also have $\mathcal{F}_j \to \{Z_j\} \notin \Sigma_j^+$ for at least one j according to Lemma 1, in particular $Z_j \notin \mathcal{F}_j$.

Now take $r = \{(X_j : t_{j1}), (X_j : t_{j2})\}$. Then for all $Y = X_1(Y_1) \oplus \cdots \oplus X_n(Y_n) \in \mathcal{Y} \subseteq \mathcal{F}$ we have $Y_j \in \mathcal{F}_j$, and we obtain

$$\pi^X_Y((X_j : t_{j1})) = (X_j : \pi^{X_j'}_{Y_j}(t_{j1})) = (X_j : \pi^{X_j'}_{Y_j}(t_{j2})) = \pi^X_Y((X_j : t_{j2})).$$

On the other hand, $Z_j \notin \mathcal{F}_j$ implies

$$\pi^X_Z((X_j : t_{j1})) = (X_j : \pi^{X_j'}_{Z_j}(t_{j1})) \neq (X_j : \pi^{X_j'}_{Z_j}(t_{j2})) = \pi^X_Z((X_j : t_{j2})).$$

That is $r \not\models \mathcal{Y} \to \{Z\}$, and hence $r \not\models \mathcal{Y} \to \mathcal{Z}$ by the soundness of the fragmentation rule (8).

We finally show $r \models \Sigma$ in both cases, which proves the theorem. We show this for the case that the outermost constructor is the not union-constructor. If the outermost constructor is the union-constructor, then according to Lemma 1 we have to show $r_j \models \Sigma_j$ for all $j = 1, \ldots, n$, the proof of which is analogous to the first case. So let $\mathcal{U} \to \mathcal{V} \in \Sigma$. We distinguish two cases:

- If $\mathcal{U} \subseteq \mathcal{F}$, then $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}$. The reflexivity axiom and the transitivity rule allow us to derive $\mathcal{Y} \to \mathcal{V} \in \Sigma^+$, which means $\mathcal{V} \subseteq \mathcal{F}$ and thus $\pi^X_V(t_1) = \pi^X_V(t_2)$ for all $V \in \mathcal{V}$, i.e. $r \models \mathcal{U} \to \mathcal{V}$.
- If $\mathcal{U} \subseteq \mathcal{F}$, then there is some $U \in \mathcal{U}$ with $\pi^X_U(t_1) \neq \pi^X_U(t_2)$, which immediately implies $r \models \mathcal{U} \to \mathcal{V}$.

□

In fact, the proof shows a bit more than claimed. We only needed that properties 4, 5(a) and 6 of Theorem 2 are immediately satisfied, because the corresponding attributes can both appear as subattributes of an attribute $X' \in \mathrm{emb}(X)$. This gives the following theorem.

Theorem 9 (Completeness Theorem for FDs). *Let $X \in \mathcal{N}$ be a nested attribute such that no subattribute $Y \in \mathcal{S}(X')$ of an embedded attribute $X' \in emb(X)$ has the form $X'_I\{\lambda\}$ with $|I| \geq 2$. Then the set of axioms and rules in Theorems 3, 5 excluding the set axiom (10) is complete for the implication of FDs on $\mathcal{S}(X)$.*

Let us now investigate the question, whether the restriction on the attribute X in Theorem 9 can be dropped. Unfortunately, this is not the case, i.e. if both the union and the set constructor are present, more precisely, if the union constructor does appear immediately inside a set constructor, then there is no finite axiomatisation.

Theorem 10. *If $X \in \mathcal{N}$ is a nested attribute such that there exists a subattribute $X'_I\{\lambda\} \in \mathcal{S}(X')$ with $|I| \geq 2$ of an embedded attribute $X' \in emb(X)$, then there does not exist a finite, sound and complete system of axioms and rules for the implication of FDs on $\mathcal{S}(X)$.*

The proof will exploit a general result about closures under k-ary implication, which was proven in [3, Proposition 9.3.2]. We first define the necessary notions for this result.

Definition 12. Let $X \in \mathcal{N}$ be a nested attribute, Γ a class of dependencies on $\mathcal{S}(X)$, and $k \geq 0$.

A set $\Sigma \subseteq \Gamma$ of dependencies on $\mathcal{S}(X)$ is *closed under implication* with respect to Γ iff $\Sigma \models \varphi$ implies $\varphi \in \Sigma$ for all $\varphi \in \Gamma$.

Σ is *closed under k-ary implication* with respect to Γ iff for all $\varphi \in \Gamma$ whenever $\Sigma' \models \varphi$ holds for some $\Sigma' \subseteq \Sigma$ with $|\Sigma'| \leq k$, then this implies $\varphi \in \Sigma$.

Furthermore, we will exploit *ground derivation rules* that result from the derivation rules we used so far by instantiating the variables in the premise and the conclusion in such a way that the side conditions are satisfied.

Theorem 11. *Let $X \in \mathcal{N}$ be a nested attribute, Γ a class of dependencies on $\mathcal{S}(X)$, and let $k \geq 0$. Then there exists a k-ary ground axiomatisation for Γ iff each $\Sigma \subseteq \Gamma$ that is closed under k-ary implication is also closed under implication.*

The proof of Theorem 11 was given in [3, Proposition 9.3.2]. In fact, the proposition was formulated for the relational model, but the proof does not depend on that.

of Theorem 10. If for the class Γ of FDs on $\mathcal{S}(X)$ we had a finite axiomatisation, then there would exist some $k \geq 0$ such that Γ has a k-ary ground axiomatisation. According to Theorem 11 $\Sigma \subseteq \Gamma$ that is closed under k-ary implication would also be closed under implication. So take $X = X\{X_1(X_1') \oplus \cdots \oplus X_{k+2}(X_{k+2}')\}$ and the set

$$\Sigma_k = \{\{X_{\{1,\ldots,k+1\}}\{\lambda\}, X_{\{i,k+2\}}\{\lambda\}\} \to \{X\} \mid i = 1, \ldots, k+1\}.$$

By looking at instances that satisfy only k of these $k+1$ FDs we see that there is no k-ary implication of $\varphi_k = \{X_{\{1,\ldots,k+1\}}\{\lambda\}\} \to \{X_{\{k+1\}}\{\lambda\}\}$ from Σ_k. So the k-ary closure of Σ_k will not contain φ_k.

On the other hand we obviously have $\Sigma_k \models \varphi_k$, so the closure of Σ_k will contain φ_k. That is, the k-ary closure of Σ_k is not closed under implication contradicting our assumption.

□

5 Extensions

In this section we extend the work on FDs and wFDs in several directions. First we will consider dependencies also on embedded attributes that were introduced in Section 3. We will see that this has very little impact on the theory, as we can show a completeness result also for these dependencies without extending the set of axioms and rules. Secondly, we will abandon the restriction on the trees to be finite and look at rational trees. Also this extension will not require additional rules.

5.1 Embedded Dependencies

The set $emb(X)$ of embedded attributes of a nested attribute X is simply characterised by: $X' \in emb(X)$ iff X' occurs somewhere within the nested structure of X. Embedded attributes are used in the proof of Theorem 2 in [28], but the theory of FDs and wFDs in the previous section did not make much further use of these attributes.

However, the lifting rules (19)-(23) implicitly contained FDs on embedded attributes. Nevertheless, we only looked at sets Σ of dependencies on $\mathcal{S}(X)$, so only trivial dependencies on embedded attributes played a role. We now make dependencies on embedded attributes explicit.

Definition 13. Let $X \in \mathcal{N}$. An *embedded functional dependency* (eFD) on $\mathcal{S}(X)$ is an expression $X' : \mathcal{Y} \to \mathcal{Z}$ with $X' \in emb(X)$ and $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X')$. An *embedded weak functional dependency* (ewFD) on $\mathcal{S}(X)$ is an expression $X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with $X' \in emb(X)$, an index set I and $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X')$.

In the following we consider again instances of X, i.e. finite sets $r = r(X) \subseteq dom(X)$. For each embedded attribute $X' \in emb(X)$, r induces an instance $r(X') \subseteq dom(X')$ in the obvious way: $v' \in r(X')$ iff there exists some $v \in r(X)$ such that v' occurs in v at the position indicated by X' in the nesting of X. Using this extension of instances, the satisfaction definition for eFDs and ewFDs is straightforward.

Definition 14. Let r be an instance of X. We say that r *satisfies the eFD* $X' : \mathcal{Y} \to \mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models X' : \mathcal{Y} \to \mathcal{Z}$) iff for all $t_1, t_2 \in r(X')$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ we also have $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$.

An instance $r \subseteq dom(X)$ *satisfies the ewFD* $X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ on $\mathcal{S}(X)$ (notation: $r \models X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$) iff for all $t_1, t_2 \in r(X')$ there is some $i \in I$ with $\{t_1, t_2\} \models \mathcal{Y}_i \to \mathcal{Z}_i$.

According to this definition we may again identify an ewFD $X' : \{\!\{\mathcal{Y} \to \mathcal{Z}\}\!\}$, i.e. the index set contains exactly one element, with an "ordinary" eFD $X' : \mathcal{Y} \to \mathcal{Z}$.

If Σ is a set of eFDs or ewFDs on $\mathcal{S}(X)$, we write again $\Sigma \models \psi$, if the eFD or ewFD ψ is implied by Σ, and $\Sigma \vdash \psi$, if the eFD or ewFD ψ can be derived from Σ by means of some set $\mathfrak{R}$ of axioms and rules. In this way we retain the definition of Σ^* and Σ^+ for a set of eFDS or ewFDs on $\mathcal{S}(X)$.

We may further introduce another extension to FDs and wFDs by means of *contexts*. A *context* is a set of embedded attributes, i.e. $C \subseteq emb(X)$. A context C is *non-trivial* for $X' \in emb(X)$ iff no $X'' \in C$ is a subattribute of X' nor can it be rewritten as a record attribute with X' as one of its components.

Definition 15. Let $X \in \mathcal{N}$. A *contextual functional dependency* (cFD) on $\mathcal{S}(X)$ is an expression $C \mid X' : \mathcal{Y} \to \mathcal{Z}$ with $X' \in emb(X)$, a non-trivial context C, and $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X')$. A *contextual weak functional dependency* (cwFD) on $\mathcal{S}(X)$ is an expression $C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with $X' \in emb(X)$, a non-trivial context C, an index set I and $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(X')$.

A context C partitions an instance $r(X)$ into disjoint instances using an equivalence relation $\sim_C$ defined as follows: $v_1 \sim_C v_2$ iff for each $Y \in C$ there exists some $v \in dom(Y)$ appearing in both v_1 and v_2 as the only value with this property. An equivalence class of $r(X)$ with respect to $\sim_C$ is called a *C-restricted fragment* of $r(X)$.

Definition 16. Let r be an instance of X. We say that r *satisfies the cFD* $C \mid X' : \mathcal{Y} \to \mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models C \mid X' : \mathcal{Y} \to \mathcal{Z}$) iff each C-restricted fragment of $r(X)$ satisfies the eFD $X' : \mathcal{Y} \to \mathcal{Z}$.

r *satisfies the cwFD* $C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ on $\mathcal{S}(X)$ (notation: $r \models C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$) iff each C-restricted fragment of $r(X)$ satisfies the ewFD $X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

5.2 Extended Completeness Result

Let us first look at the derivation rules in Theorems 3, 5 and 6. In all these rules except the lifting rules (19)-(23) all dependencies are defined on $\mathcal{S}(X)$ with X left implicit, and the soundness proofs use arbitrary instances of X. In making X explicit, we turn the rules into derivation rules for eFDs and ewFDs. We can even turn them into derivation rules for cFDs and cwFDs by adding the prefix $C \mid X$ to all occurring dependencies. The soundness proof remains in all cases the same, because the notion of satisfaction defined in Definitions 14 and 16 only requires to consider only specific instances of X.

For the lifting rules the situation is similar; the only difference is that the dependencies in the rule conclusions are defined on some attribute X, while those in the premises are defined on some $X' \in emb(X)$. More precisely, we have $X = \{X'\}$, $X = X(X_1, \ldots, X_n)$ with $X_i = X'$, $X = X_1(X_1') \oplus \cdots \oplus X_n(X_n')$ with $X_i' = X'$, $X = \langle X' \rangle$, and $X = [X']$, respectively. Nevertheless, making these attributes explicit and adding a context prefix defines derivation rules for eFDs, ewFDs, cFDs and cwFDs. These rules are obviously sound, as the soundness proof (Theorem 5) does not require any change except the mentioned syntactic modifications.

For cwFDs (and hence also for cFDs) we can add another derivation rule linking different contexts together. For this we define that context C' is *more restrictive* than context C (notation: $C' \trianglelefteq C$) iff $\sim_{C'} \subseteq \sim_C$ holds. Obviously, the empty context is the least restrictive one. Then we get the following derivation rule (*context rule*):

$$\frac{C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}}{C' \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}} \, C' \trianglelefteq C \tag{34}$$

Theorem 12. *The rules in Theorems 3, 5 and 6 (with the syntactic modifications above) and rule (34) are sound for the implication of eFDs, ewFDs, cFDs and cwFDs.*

Proof. We only have to show the soundness of the context rule (34). So assume that $r = r(X)$ satisfies $C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$. Take a C'-restricted fragment $r_{C'}(X)$ of $r(X)$ and $t_1, t_2 \in r_{C'}(X')$. As $\sim_{C'} \subseteq \sim_C$ holds, t_1 and t_2 must be in the same C-restricted fragment $r_C(X')$ of $r(X)$. Furthermore, there is some $i \in I$ such that $\{t_1, t_2\}$ satisfies the FD $\mathcal{Y}_i \to \mathcal{Z}_i$, hence $r \models C' \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$. □

Now we can even extend the completeness result in Theorem 7 to cwFDs.

Theorem 13 (Completeness Theorem). *The set of axioms and rules in Theorems 3, 5 and 6 (with the syntactic modifications above) and rule (34) is complete for the implication of cwFDs on $\mathcal{S}(X)$.*

Proof. Let Σ be a set of cwFDs on $\mathcal{S}(X)$ and assume $C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^+$. Due to the context rule also $\emptyset \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^+$ holds, so we actually have to deal with an ewFD. Due to the union rule (7) we must have $X' : \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ for some selected $Z_i \in \mathcal{Z}_i$. Furthermore, due to the

left union rule (25) we get $X' : \{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \notin \Sigma^+$ with $\mathcal{Y} = \bigcup_{i \in I} \mathcal{Y}_i$. Due to the lifting rules we may assume that there is no cwFD $C' \mid X'' : \{\!\{\mathcal{Y}'_j \to \mathcal{Z}'_j \mid j \in J\}\!\} \in \Sigma$ with $X'' \in emb(X')$; otherwise we could use a restricted set of dependencies.

Let $\mathcal{Z} = \{Z \mid Z \geq Z_i \text{ for some } i \in I\}$ and $\mathcal{U} = \mathcal{S}(X') - \mathcal{Y} - \mathcal{Z}$. Due to the reflexivity axiom (1) we obviously have $Z_i \notin \mathcal{Y}$, and then $\mathcal{Y} \sqcap \mathcal{Z} = \emptyset$ due to the subattribute axiom (2). Due to the shift rule (26) there must exist some $\mathcal{U}' \subseteq \mathcal{U}$ with $\{\!\{\mathcal{Y} \cup \mathcal{U}' \to \{Z\} \mid Z \in \mathcal{Z} \cup (\mathcal{U} - \mathcal{U}')\}\!\} \notin \Sigma^+$. Otherwise we could derive $X' : \{\!\{\mathcal{Y} \to \{Z\} \mid Z \in \mathcal{Z}\}\!\}$, and thus $X' : \{\!\{\mathcal{Y} \to \{Z_i\} \mid i \in I\}\!\} \in \Sigma^+$ contradicting our assumption.

Let $\mathcal{U}'$ be maximal with the given property. Then using Lemma 2 we obtain that $\mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$ is a coincidence ideal.

Without loss of generality we may assume $X' \neq X$; otherwise we are back to the case that was already handled in the proof of Theorem 7. Therefore, due to the restructuring rules we can assume that the outermost constructor in X' is not the union constructor. Then we can apply the Central Theorem 2, which gives us $r' = \{t'_1, t'_2\} \subseteq dom(X')$ with $\pi^X_Y(t'_1) = \pi^X_Y(t'_2)$ iff $Y \in \mathcal{F} = \mathcal{Y} \cup \mathcal{U}'$. In particular, $\pi^X_Y(t'_1) = \pi^X_Y(t'_2)$ for all $i \in I$ and $Y \in \mathcal{Y}_i$, and $\pi^X_{Z_i}(t'_1) \neq \pi^X_{Z_i}(t'_2)$ for all $i \in I$. That is, $r' \not\models \{\!\{\mathcal{Y}_i \to \{Z_i\} \mid i \in I\}\!\}$. From the soundness of the fragmentation rule (8) we conclude $r' \not\models \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

We now "lift" r' to an instance r of X such that $r(X') = r'$ holds. For this take a chain $X_0, \ldots, X_k$ of maximal length with $X_0 = X$, $X_k = X'$, and $X_i \in emb(X_{i-1}) - \{X_i\}$ for $i = 1, \ldots, k$. Then for $i = k, \ldots, 0$ define inductively $t_{i1}, t_{i2} \in dom(X_i)$ starting with $t_{kj} = t'_j$ for $j = 1, 2$.

- For $X_i = X_i(X'_1, \ldots, X'_\ell)$ define $t_{ij} = (t^1_{ij}, \ldots, t^\ell_{ij})$ with

$$t^h_{ij} = \begin{cases} t_{(i+1)j} & \text{if } X'_h = X_{i+1} \\ \tau^{X'_h}_\lambda & \text{else} \end{cases}.$$

- For $X_i = X_i\{X_{i+1}\}$ define $t_{ij} = \{t_{(i+1)j}\}$, if $X_i \notin C$ holds; otherwise take $t_{i1} = t_{i2} = \{t_{(i+1)1}, t_{(i+1)2}\}$.
- For $X_i = X_i\langle X_{i+1}\rangle$ define $t_{ij} = \langle t_{(i+1)j}\rangle$, if $X_i \notin C$ holds; otherwise take $t_{i1} = t_{i2} = \langle t_{(i+1)1}, t_{(i+1)2}\rangle$.
- For $X_i = X_i[X_{i+1}]$ define $t_{ij} = [t_{(i+1)j}]$, if $X_i \notin C$ holds; otherwise take $t_{i1} = t_{i2} = [t_{(i+1)1}, t_{(i+1)2}]$.

Finally, take $t_j = t_{0j}$ for $j = 1, 2$ and $r = \{t_1, t_2\}$. Then obviously $r(X') = r'$ holds. Consequently, $r \not\models X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

If C contains an attribute $\bar{X}$ with $X' \in emb(\bar{X})$, then r only contains one element, so it is its only C-restricted fragment. In this case we obviously get $r \not\models C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$. However, if C does not contain such an attribute, then due to our construction we also get that r equals its only C-restricted fragment, hence again $r \not\models C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$.

Now take $C' \mid X'' : \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\} \in \Sigma$. If $X' \notin emb(X'')$ holds, then due to our construction $r(X'')$ will only contain one element, hence r trivially satisfies this cwFD. Thus we can assume $X' \in emb(X'')$. In this case we unnest the attributes in $\mathcal{V}_j$ and $\mathcal{W}_j$, until we obtain $\mathcal{V}'_j.\mathcal{W}'_j \subseteq \mathcal{S}(X')$. As in Lemma 3 we consider two cases:

1. If $\mathcal{V}'_j \not\subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$, we get $\pi^{X'}_{V'}(t'_1) \neq \pi^{X'}_{V'}(t'_2)$ for some $V' \in \mathcal{V}'_j$. We can assume that $r(X'')$ contains two elements, say $r(X'') = \{t''_1, t''_2\}$. Then we get $\pi^{X''}_{V}(t''_1) \neq \pi^{X''}_{V}(t''_2)$ for some $V \in \mathcal{V}_j$. Thus $r \models X'' : \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$, hence also $r \models C' \mid X'' : \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$.

2. If $\mathcal{V}'_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for all $j \in J$, then using the same arguments as in the proof of Lemma 3 we must have $\mathcal{W}'_j \subseteq \mathcal{Y} \cup \mathcal{U}'$ for some $j \in J$. By construction of r' we get $\pi^{X'}_{W'}(t'_1) = \pi^{X'}_{W'}(t'_2)$ for all $W' \in \mathcal{W}'_j$. Due to our construction oif r this implies $\pi^{X''}_{W}(t''_1) = \pi^{X''}_{W}(t''_2)$ for all $W \in \mathcal{W}_j$. Thus $r \models X'' : \mathcal{V}_j \to \mathcal{W}_j$. This implies $r \models C' \mid X'' : \{\!\{\mathcal{V}_j \to \mathcal{W}_j \mid j \in J\}\!\}$ due to the soundness of the weakening rule and the context rule.

So $r \models \Sigma$, hence also $r \models \Sigma^*$, from which we get $C \mid X' : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\} \notin \Sigma^*$. This completes the proof of the theorem.

□

Note that this completeness proof is only a slight modification of the proof of Theorem 7 exploiting the additional context rule, while the major arguments remain the same. Therefore, it is straightforward to apply these modifications also to the proof of Theorem 9, which leads to the following theorem on the completeness of cFDs (with the necessary syntactic modifications of the rules).

Theorem 14. *Let $X \in \mathcal{N}$ be a nested attribute such that no subattribute $Y \in \mathcal{S}(X')$ of an embedded attribute $X' \in emb(X)$ has the form $X'_I\{\lambda\}$ with $|I| \geq 2$. Then the set of axioms and rules in Theorems 3, 5 excluding the set axiom (10) together with the context rule (34) is complete for the implication of cFDs on $\mathcal{S}(X)$.*

5.3 Rational Trees

So far, all nested attributes had a fixed depth, and all complex values were representable as finite trees. In order to capture object oriented structures as in [30] and XML as in [1], we have to allow recursively defined attributes that take *rational trees* as their values, i.e. trees with only finitely many distinct subtrees. The notion of nested attributes has already been extended in this direction in [19]; we simply have to add $\mathcal{L} \subseteq \mathcal{N}$ to Definition 2 of nested attributes.

Definition 17. Let $\mathcal{U}$ be a universe and $\mathcal{L}$ a set of labels. The set $\mathcal{N}$ of *nested attributes* (over $\mathcal{U}$ and $\mathcal{L}$) is the smallest set with $\lambda \in \mathcal{N}$, $\mathcal{U} \subseteq \mathcal{N}$, $\mathcal{L} \subseteq \mathcal{N}$, and satisfying the following properties:

- for $X \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X(X'_1, \ldots, X'_n) \in \mathcal{N}$;

- for $X \in \mathcal{L}$ and $X' \in \mathcal{N}$ we have $X\{X'\} \in \mathcal{N}$, $X[X'] \in \mathcal{N}$, and $X\langle X'\rangle \in \mathcal{N}$;
- for $X_1, \ldots, X_n \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X_1(X'_1) \oplus \cdots \oplus X_n(X'_n) \in \mathcal{N}$.

We say that a label $Y \in \mathcal{L}$ occurring inside a nested attribute X, is a *defining label* iff it is introduced by one of the three cases in Definition 2. Otherwise it is a *referencing label*. We require that each label Y appears at most once as a defining label in a nested attribute X, and that each referencing label also occurs as a defining label. In other words, if we represent a nested attribute by a labelled tree, a defining label is the label of a non-leaf node, and a referencing label is the label of a leaf node.

We still have to extend Definition 3. For this assume $X \in \mathcal{N}$ and let Y be a referencing label in X. If we replace Y by the nested attribute that is defined by Y within X, we call the result an *expansion* of X. Note that in such an expansion a label may now appear more than once as a defining label, but all the nested attributes defined by a label can be identified, as the corresponding sets of expansions are identical.

In order to define domains assume set of *label variables* $\psi(Y)$ for each $Y \in \mathcal{L}$. Then for each expansion X' of a nested attribute X we define $dom(X')$ as in Definition 3 with the following modifications:

- for a referencing label Y we take $dom(Y) = \psi(Y)$;
- for a label Y defining the nested attribute Y' take $dom(Y) = \{y : v \mid y \in \psi(Y), v \in dom(Y')\}$;
- allow only such values v in $dom(X')$, for which the values of referencing labels also occur inside v exactly once at the position of a defining label.

Finally, define $dom(X) = \bigcup_{X'} dom(X')$, where the union spans over all expansions X' of X.

There is no need to change the definition of subattributes. We only have to be aware of the fact that now a nested attribute has several expansions, and they all can be used to define subattributes. Also the definitions of FDs and wFDs do not require more than the tiny addition that the sets of subattributes used in them must be finite (which they were automatically so far).

With these modifications we can easily repeat the whole theory of coincidence ideals and dependencies. The decisive property we exploit is the finiteness of a set Σ of wFDs. Then we can always find an expansion of X that is large enough such that the remaining referencing labels can actually be treated in the same way as simple attributes. In particular, the domain associated with these labels is infinite. This leads immediately to the following result.

Theorem 15. *The soundness and completeness theorems 3, 5, 6, 7 and 9 also hold for nested attributes X with the extensions from Definition 17.*

The same arguments also apply to embedded and contextual FDs and wFDs. We only have to be careful with the notation of embedded attributes in their definition, as these are no longer unique. Thus, instead of $X' \in emb(X)$ we consider *embedding paths* $X_0, \ldots, X_k$ of maximal length with $X_0 = X$, $X_k = X'$ and $X_i \in emb(X_{i-1}) - \{X_i\}$ for $i = 1, \ldots, k$. We also define $\mathcal{S}(X_0, \ldots, X_k) = \mathcal{S}(X_k)$ as the associated set of subattributes.

Definition 18. Let $X \in \mathcal{N}$. An *embedded functional dependency* (eFD) on $\mathcal{S}(X)$ is an expression $P : \mathcal{Y} \to \mathcal{Z}$ with an embedding path P and $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(P)$. An *embedded weak functional dependency* (ewFD) on $\mathcal{S}(X)$ is an expression $P : \{\!\{\mathcal{Y}_i \to \mathcal{Z}_i \mid i \in I\}\!\}$ with an embedding path P, an index set I and $\mathcal{Y}_i, \mathcal{Z}_i \subseteq \mathcal{S}(P)$.

This definition carries over naturally to contextual dependencies. Using the same argument as for wFDs we can also generalise the soundness and completeness results for contextual dependencies.

Theorem 16. *The soundness and completeness theorems 12, 13 and 14 also hold for nested attributes X with the extensions from Definition 17.*

6 Related Work

Apart from previous work by us and our colleagues Link and Hartmann that has been intensively used in this article there are two major related research groups working on dependencies on trees. Both Arenas and Libkin (see [5]) and Vincent, Liu and Liu (see [37]) place their work directly in the context of XML, while we take a more general approach using various constructors and rational trees. This implies that depending on the choice of incorporating order or not, these related approaches only handle one of the three bulk constructors, either lists or sets, while we take all three into account simultaneously. In fact, both Arenas and Libkin and Vincent et al. do not consider order, so the related case in our work refers to the use of the set constructor, apparently exactly the case, for which FDs cannot be finitely axiomatised. Furthermore, none of the other groups handles weak functional dependencies.

As emphasised in [37], but not proven, the different notions of XML FDs in the work by Arenas and Libkin and Vincent et al., respectively, coincide in case of complete information. Vincent, Liu and Liu claim that their notion of FDs actually captures incomplete information, while Arenas's and Libkin's work does. In our work, incomplete information is captured by the null attribute λ, so it boils down to the question, whether our definition of FDs can capture those defined by the other groups.

As emphasised in Section 5 the notion of FD from Definition 9 is bound to finite trees of fixed depth, while the work by the others deal with the variable depth of XML trees. So, without the extension to rational trees our notion of FDs cannot capture the other ones nor vice versa, because our definition of FDs involves complex subattributes, so equality is "generated" even on sets. However, taking

cFDs on rational trees, it is not too difficult to see that the XFDs defined in [5] are actually representable in our framework. We may always restrict ourselves to XFDs $p_1 \ldots p_k \to p$, i.e. the right hand side is a singleton. Then the right hand side defines an embedded attribute X', while the paths on the left hand side then give rise to either a subattribute of X' or the context subattributes. We illustrate this relation by a final example referring to the DTD in [5, Example 1.1] and the XFDs in [5, Example 4.1].

Example 2. The DTD in [5, Example 1.1] can be represented by the nested attribute

$$courses\{course(CNO, title(S), taken\_by\{student(SNO, name(S), grade(S))\})\}.$$

Then the following eFDs and cFDs represent the XFDs in [5, Example 4.1]:

$$course : \{course(CNO)\} \to \{course(CNO, title(S), taken\_by\{student(SNO, name(S), grade(S))\})\}$$
$$course \mid student : \{student(SNO)\} \to \{student(SNO, name(S), grade(S))\}$$
$$student : \{student(SNO)\} \to \{student(name(S))\}$$

7 Conclusions

In this article we completed our work on the axiomatisation of functional dependencies and weak functional dependencies on trees with restructuring. These trees arise from constructors for complex values comprising arbitrarily nesting of finite sets, multisets, lists, disjoint unions and records and a "null" attribute. Restructuring, i.e. non-trivial equivalence between these attributes are mainly due to the presence of the union constructor. While our previous work in [27] captured the case, where so called counter-attributes were excluded, we now were able to provide a sound and complete set of derivation rules for weak functional dependencies without this restriction. The price for this result was a very deep and very technical investigation of certain ideals in the algebra of subattributes leading to the central theorem on coincidence ideals, which gives an exact characterisation of sets of subattributes, on which two complex values coincide. We were further able to generalise the axiomatisation to capture dependencies on embedded attributes thereby including classes of FDs defined by others (see e.g. [5]).

Though our results require quite a heavy mathematical machinery, the technical characterisation of coincidence ideals in [28] to remove a seemingly not severe restriction in our previous results, we should emphasise that the unrestricted classes of FDs and wFDS treated in this article capture counting by means of subattributes. That is, whenever we have a multiset or list attribute, the projection of a complex value to a counter-attribute tells us how many values of a certain kind appear in this multiset or list. This is a concept that has not been handled in the context of functional dependencies before.

Unfortunately, for set attributes this is slightly different, as the counter-attributes in this case merely function as flags indicating, whether the subset of values of a certain kind is empty or not. This shows us that there is still more work needed to capture counting completely. In [29] we started work in this direction by deliberately adding more restructuring rules – so far, only intrinsic, unavoidable equivalences have been used. However, we may even take a list and forget the order of its elements, thus mapping it to a multiset, or map a multiset to its set of elements, i.e. we obtain an extension of the subattribute order by adding $X[Y] \geq X\langle Y\rangle \geq X\{Y\}$. Similarly, we could treat a set attribute as a multiset attribute, and then define FDs on it by using the subattributes of this corresponding multiset attribute.

The work in [29] only contains the first step in this direction, as only functional dependencies not involving the union constructor are handled. That is, the more interesting counter-attributes and the intrinsic restructuring rules are absent. The natural question is, how our results in this article can be generalised to deal also with these extensions to restructuring in general. Other open problem to be addressed in future are linked to other classes of dependencies, e.g. multi-valued and join dependencies as in [21] and [40] and to the existence of Armstrong instances (see e.g. [27]).

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, 2000.

[2] S. Abiteboul and R. Hull. Restructuring hierarchical database objects. *Theoretical Computer Science*, 62(1-2):3–38, 1988.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[4] M. Arenas and L. Libkin. A normal form for XML documents. In *PODS 2002.* ACM, 2002.

[5] M. Arenas and L. Libkin. A normal form for XML documents. *ACM Transactions on Database Systems*, 29(1):195–232, 2004.

[6] W. W. Armstrong. Dependency structures of database relationships. *Information Processing*, pages 580–583, 1974.

[7] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach.* Benjamin Cummings, 1992.

[8] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *Tenth WWW Conference.* IEEE, 2001.

[9] P. P. Chen. The Entity-Relationship model: Towards a unified view of data. *ACM Transactions Database Systems*, 1:9–36, 1976.

[10] P. P. Chen. English sentence structure and Entity-Relationship diagrams. *Information Science*, 29:127–149, 1983.

[11] J. Demetrovics and G. Gyepesi. On the functional dependency and some generalizations of it. *Acta Cybernetica*, 5:295–305, 1981.

[12] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *PODS 2001*. ACM, 2001.

[13] W. Fan and J. Siméon. Integrity constraints for XML. In *PODS 2000*. ACM, 2000.

[14] S. Hartmann. Decomposing relationship types by pivoting and schema equivalence. *Data & Knowledge Engineering*, 39:75–99, 2001.

[15] S. Hartmann and S. Link. On functional dependencies in advanced data models. *Electronic Notes in Theoretical Computer Science*, 84, 2003.

[16] S. Hartmann, S. Link, and K.-D. Schewe. Generalizing Boyce-Codd normal form to conceptual databases. In *Information Modelling and Knowledge Bases XV*, volume 105 of *Frontiers in Artificial Intelligence and Applications*, pages 88–105. IOS Press, 2004.

[17] S. Hartmann, S. Link, and K.-D. Schewe. Reasoning about functional and multi-valued dependencies in the presence of lists. In D. Seipel and J. M. Turull Torres, editors, *Foundations of Information and Knowledge Systems*, volume 2942 of *LNCS*, pages 134–154. Springer Verlag, 2004.

[18] S. Hartmann, S. Link, and K.-D. Schewe. Weak functional dependencies in higher-order datamodels. In D. Seipel and J. M. Turull Torres, editors, *Foundations of Information and Knowledge Systems*, volume 2942 of *LNCS*, pages 116–133. Springer Verlag, 2004.

[19] S. Hartmann, S. Link, and K.-D. Schewe. Functional dependencies over XML documents with DTDs. *Acta Cybernetica*, 17(1):153–171, 2005.

[20] S. Hartmann, S. Link, and K.-D. Schewe. Axiomatisation of functional dependencies in the presence of records, lists, sets and multisets. *Theoretical Computer Science*, 355:167–196, 2006.

[21] S. Hartmann, S. Link, and K.-D. Schewe. Functional and multi-valued dependencies in nested databases generated by record and list constructor. *Annals of Mathematics and Artificial Intelligence*, 46:111–164, 2006.

[22] R. Hull and R. King. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3), 1987.

[23] W. Y. Mok, Y. K. Ng, and D. W. Embley. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21:77–106, 1996.

[24] Z. M. Özsoyoglu and L. Y. Yuan. A new normal form for nested relations. *ACM Transactions on Database Systems*, 12:111–136, 1987.

[25] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model.* Springer-Verlag, 1989.

[26] A. Sali. Minimal keys in higher-order datamodels. In D. Seipel and J. M. Turull Torres, editors, *Foundations of Information and Knowledge Systems*, volume 2942 of *LNCS*, pages 242–251. Springer Verlag, 2004.

[27] A. Sali and K.-D. Schewe. Counter-free keys and functional dependencies in higher-order datamodels. *Fundamenta Informaticae*, 70(3):277–301, 2006.

[28] A. Sali and K.-D. Schewe. A characterisation of coincidence ideals for complex values. *Journal of Universal Computer Science*, 15(1):304–354, 2009.

[29] K.-D. Schewe. Functional dependencies with counting on trees. *Journal of Universal Computer Science*, 11(12):2063–2075, 2005.

[30] K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(4):49–85, 1993.

[31] Z. Tari, J. Stokes, and S. Spaccapietra. Object normal forms and dependency constraints for object-oriented schemata. *ACM Transactions on Database Systems*, 22:513–569, 1997.

[32] B. Thalheim. *Dependencies in Relational Databases.* Teubner-Verlag, 1991.

[33] B. Thalheim. Foundations of entity-relationship modeling. *Annals of Mathematics and Artificial Intelligence*, 6:197–256, 1992.

[34] B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology.* Springer-Verlag, 2000.

[35] A. M. Tjoa and L. Berger. Transformation of requirement specifications expressed in natural language into an EER model. In *Entity-Relationship Approach*, volume 823 of *LNCS*. Springer-Verlag, 1993.

[36] M. Vincent. *The semantic justification for normal forms in relational database design.* PhD thesis, Monash University, Melbourne, Australia, 1994.

[37] M. Vincent, J. Liu, and C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM Transactions on Database Systems*, 29(3):445–462, 2004.

[38] M. W. Vincent and J. Liu. Functional dependencies for XML. In *Web Technologies and Applications: 5th Asia-Pacific Web Conference*, volume 2642 of *LNCS*, pages 22–34. Springer-Verlag, 2003.

[39] M. W. Vincent and J. Liu. Multivalued dependencies and a 4NF for XML. In *Advanced Information Systems Engineering: 15th International Conference CAiSE 2003*, volume 2681 of *LNCS*, pages 14–29. Springer-Verlag, 2003.

[40] M. W. Vincent and J. Liu. Multivalued dependencies in XML. In *British National Conference on Database Systems: BNCOD 2003*, volume 2712 of *LNCS*, pages 4–18. Springer-Verlag, 2003.

[41] J. Wang and R. Topor. Removing XML data redundancies using functional and equality-generating dependencies. In G. Dobbie and H. Williams, editors, *Database Technologies 2005 - Sixteenth Australasian Database Conference*, volume 39 of *CRPIT*, pages 65–74. Australian Computer Society, 2005.

Received 11th September 2009

Acta Cybernetica **20** (2011) 331–346.

Mining High Utility Itemsets in Massive Transactional Datasets*

Vu Duc Thi† and Nguyen Huy Duc‡

Abstract

Mining High Utility Itemsets from a transaction database is to find itemsets that have utility beyond an user-specified threshold. Existing High Utility Itemsets mining algorithms suffer from many problems when being applied to massive transactional datasets. One major problem is the high memory dependency: the gigantic data structure built is assumed to fit in the computer main memory. This paper proposes a new disk-based High Utility Itemsets mining algorithm, which achieves its efficiency by applying three new ideas. First, transactional data is converted into a new database layout called Transactional Array that prevents multiple scanning of the database during the mining phase. Second, for each frequent item, a relatively small independent tree is built for summarizing co-occurrences. Finally, a simple and non-recursive mining process reduces the memory requirements as minimum candidacy generation and counting is needed. We have tested our algorithm on several very large transactional databases and the results show that our algorithm works efficiently.

Keywords: High Utility Itemset Mining, COUI-tree

1 Introduction

A framework for high utility itemset mining was proposed recently by Yao et al (H. Yao and H. J. Hamilton, 2006) [6]. In this, the value of one item is a number (the quantity of the sold item, we can call it an objective value), otherwise, it has a utility table that contains utility of all items in the dataset (we can call it a subjective value, determined by manager). Utility of a itemset is the sum of all utility of all items in that itemset. The high utility itemset mining problem is to

*This work was funded by the Vietnam's National Foundation for Science and Technology Development (NAFOSTED) via a research grant for fundamental sciences, grant number: 102.01-2010.09

†Institute of Information Technology. Viet Namese Academy of Science and Technology., E-mail: vdthi@ioit.ac.vn

‡Faculty of Information and Computer, National Training College for Teachers, Ha Noi, Viet Nam., E-mail: ducnghuy@yahoo.com

find all itemsets that have utility larger than a user specified value of minimum utility.

In [6], H. Yao and H. J. Hamilton proposed a mining method and described pruning strategies based on the mathematical properties of utility constraints. They also developed an algorithm named Umining and another heuristic based algorithm Umining_H to discover high utility itemsets.

Recent research has focused on efficient high utility mining algorithms using intermediate anti-monotone measures for pruning the search space. In [7], Liu et al. (Y. Liu, Liao, & Choudhary, 2005) propose a two phase algorithm to mine high utility itemsets. They use a transaction weighted utility (TWU) measure in the first phase to find the supersets of high utility itemsets, followed by a rescan of the database to determine the actual high utility itemsets among them. However, their algorithm is based on the candidate generation-and-test approach and so suffers from poor performance when mining dense datasets and long patterns in the same way as the Apriori algorithm for frequent pattern mining.

In this paper, we propose an efficient algorithm for utility mining in massive datasets. This algorithm rearranges database and saves it the external memory, in mining process only a small part of data is put into the internal memory and mining is based on the idea of COFI-tree algorithm by Mohammad El-Hajj and Osmar R. Zaiane presented in 2003. After all data is rearranged and stored in the external memory, we can mine high utility itemsets with a different threshold without reorganizing the database.

The rest of the paper is organized as follows: In Section 2, we define the relevant terms. Section 3 summarizes the COFI-tree algorithm used in mining frequent patterns. Section 4 describes our new algorithm for mining high utility itemsets in large datasets. The performance studies of the algorithm are given in Section 5. Section 6 contains the conclusions of the paper.

2 High Utility Itemset Mining

In this Section, we give the basic notations and the definitions of terms to describe high utility itemset mining, based on (H. Yao and H. J. Hamilton, 2006) [6]. Let $I = \{i_1, \ldots, i_n\}$ be a set of items. A transaction T is a subset of $I, T \subseteq I$. $DB = \{T_1, \ldots, T_m\}$ is a transaction database. Each transaction is assigned by an ID called TID. A subset $X \subseteq I$ which contains k different items is called k-itemset. Transaction T contains X if $X \subseteq T$.

Definition 1. *The value of item i_p in transaction T_q (at column i_p row T_q of database) is an objective value denoted as (i_p, T_q).*

Definition 2. *Calling value, which is assigned by manager for item i_p in database, based on estimating utility gaining from one unit of that item is a called subjective value denoted as $s(i_p)$.*

Normally, the subjective value is given in a table called the utility table. For

example, in Table 1 and 2, the objective value of item B at transaction T_2 is $o(B, T_2) = 12$, the subjective value of item B is $s(B) = 5$.

Definition 3. *Let x be an objective value and y be a subjective value of one item. Function $f(x, y) : R \times R \longrightarrow R$ is called the utility function calculated as follow: $f(x, y) = x \times y$.*

Table 1: Transactional database

TID	A	B	C	D	E
$T1$	0	12	2	0	2
$T2$	0	12	0	2	1
$T3$	2	0	1	0	1
$T4$	1	0	0	2	1
$T5$	0	0	4	0	2
$T6$	1	2	0	0	0
$T7$	0	20	0	2	1
$T8$	3	0	25	6	1
$T9$	1	2	0	0	0
$T10$	0	0	16	0	1

Table 2: Utility table

Item	Profit ($/unit)
A	3
B	5
C	1
D	3
E	5

Definition 4. *Let $f(x, y)$ be an utility function. The utility of item i_p in transaction T_q (denoted as $u(i_p, T_q)$) is the value of $f(x, y)$ at $o(i_p, T_q)$ and $s(i_p)$, that is $u(i_p, T_q) = f(o(i_p, T_q), s(i_p))$.*

Definition 5. *Let X be an itemset in transaction T_q. Utility of X in transaction T_q, denoted as $u(X, T_q)$, is defined as: $u(X, T_q) = \sum_{i_p \in X \subseteq T_q} u(i_p, T_q)$.*

An itemset X has an associated set of transactions in DB, denoted as db_X , where $db_X = \{T_q : X \subseteq T_q, T_q \in DB\}$.

Definition 6. *Utility of itemset X in database DB, denoted as $u(X)$, is utility sum of X itemset at all transactions of db_X, that is: $u(X) = \sum_{T_q \in db_X} u(X, T_q)$ $= \sum_{T_q \in db_X} \sum_{i_p \in X} u(i_p, T_q)$.*

For example, in Table 1 and 2, $u(B, T_2) = 12 \cdot 5 = 60$. Consider $X = \{B, D\}$, $u(X, T_2) = u(B, T_2) + u(D, T_2) = 12 \cdot 5 + 2 \cdot 3 = 66$, there are two transactions T_2 and T_7 which contain itemset X, so $db_X = \{T_2, T_7\}, u(X) = u(X, T_2) + u(X, T_7) = 172$.

Definition 7. *Given a minutil (> 0) and a itemset X. X is called high utility itemset if $u(X) \geq$ minutil; otherwise, X is called low utility itemset.*

Definition 8. *Given a transaction database DB and a minutil. The problem of mining high utility itemsets is to find HU set such that it contains all high utility itemsets, i.e.:*

$$HU = \{X : X \subseteq I, u(X) \geq minutil\}.$$

The problem of mining frequent itemsets can be seen as a special case of mining high utility itemsets when all items have the objective value of 0 or 1 and subjective value of 1. The main property used for mining frequent itemsets is Apriori. The Apriori property states that all nonempty subsets of a frequent itemset must also be frequent. It is not hard to see that this property is not correct in the case of utility. For example, in database of Table 1, we have, $u(BC) = 62 < 72 = u(BCE)$, while $u(BC) = 62 > 0 = u(BCD)$. The following section will present the fundamental idea of IM algorithm [9] for mining frequent itemsets using COFI-tree structure.

3 Mining frequent itemsets based on the structure of COFI-tree.

In 2003, Mohammad El-Haj and Osmar R. Zaiane in Department of Computing Science University of Alberta Edmonton, AB, Canada proposed IM (Inverted Matrix) algorithm [9] for mining frequent itemsets in large databases.

IM algorithm can be divided into two phases:

Phase 1: (pre-processing) It rearranges data into matrix and saves this matrix in the external memory.

Phase 2: This phase is mining matrix by using COFI-tree (Co-Occurrence Frequent Item Tree) for each item [8].

In the first phase, the Inverted Matrix is a disk-based data layout made of two parts: the index and the transactional arrays. The index contains the items and their respective frequencies. The transactional array is a set of rows in which each row is associated with one item in the index part. Each row is made of a pairs of pointers holding following information: the physical address in the index part of the next item in the same transaction, and the physical address in the row of the next item in the same transaction. Building the Inverted Matrix is accomplished in two passes of the database during the pre-processing phase. The first pass scans the whole database to find the frequency of each item. The item list is then ordered in ascending order according to their frequency. The second pass reads each transaction from the database and also orders it into ascending order based on the frequency of each item. In the index part, the location of the first item in the transaction is sought and an entry to its transactional array is added that holds

the location of the next item in this transaction. For the second item, the same process is applied, in which an entry in the transactional table of the second item is added to hold the location of the third item in the transaction. The process is then repeated for all items in this transaction. The following transaction is read next and the same applies to all of its items. This process repeats for all transactions in the database.

In the second phase, it mines the data matrix (transactional array) by using the structure of COFI-tree. It traverses the index part and ignores all non-frequent items, with each frequent item, it reads all transactions that contain the items and build a COFI-tree for this item, after that it mines all frequent itemsets in this tree. The trees are discarded as soon as mining ends and exactly the same process is repeated for other items.

COFI-tree of one item is a tree constructed by this item and all the others that have frequencies equal or greater than of that item. Each tree has a header table which contains a collection of frequent items, these items in header table are also sorted in ascending order of their frequency. Each entry in the header table have three data fields: item's name, local frequency in the COFI-tree and a pointer pointing to the first and the same item in the tree. A link list is maintained between all positions of those items in the tree. Each node of COFI-tree contains 4 data fields: item's name, S (it's support), P (it's participation, this field keeps track of how many times this item participates in a candidate generation), pointers pointing to the next same label node or null if not. More details of the algorithm could found in [8, 9].

4 Mining high utility itemsets in large dataset

Liu et al. (Y. Liu, Liao, & Choudhary, 2005) proposed the concepts of Transaction Utility (TU) and Transaction Weighted Utility (TWU) to prune the search space for mining high utility itemsets. Transaction Utility of a transaction, denoted $tu(T_q)$ is the sum of the utilities of all items in T_q, $tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$. Transaction Weighted Utility of an itemset X, denoted as $twu(X)$ is the sum of the transaction utilities of all the transactions containing X, $twu(X) = \sum_{T_q \in DB \wedge X \subseteq T_q} tu(T_q)$.

For example, in Table 1 and 2, $tu(T_2) = 12 \cdot 5 + 2 \cdot 3 + 1 \cdot 5 = 71$.

Let $X = DE$, $db_X = \{T_2, T_4, T_7, T_8\}, twu(X) = tu(T_2) + tu(T_4) + tu(T_7) + tu(T_8) = 253$.

Note: $u(X, T_q) \leq tu(T_q)$

$\Rightarrow u(X) = \sum_{T_q \in DB \wedge X \subseteq T_q} u(X, T_q) \leq \sum_{T_q \in DB \wedge X \subseteq T_q} tu(T_q) = twu(X)$.

Consider $twu(X)$ as the upper bound of $u(X)$. If X is a high utility itemset for a threshold *minutil*, then X is also a high utility TWU because $twu(X) \geq u(X) >$ *minutil*. Vice versa, if X is not a high utility TWU then X is also not a high utility itemset.

TWU-utility constraint has anti-monotone property [7], i.e.: All itemsets that contain a low utility TWU itemset ($twu(X) <$ *minutil*) is a low utility itemset. So, if X is a low utility TWU itemset ($twu(X) <$ *minutil*), X and all itemsets

that contain it are low utility itemsets, and could be removed while mining high utility itemsets.

Based on this idea, we propose a new COUI-Mine algorithm (Co-Occurrence Utility Item Mine) for mining high utility itemsets in large datasets. This algorithm can be divided into 2 phases:

Phase 1: Construct transactional array and saves it in the external memory.

Phase 2: Mining high utility itemsets by using the structure of COUI-tree.

4.1 Construct transactional array:

The algorithm separates disk-based data into two parts: the index and the transactional array. Each entry in index part contains 5 data fields: item's name, quantity, profit of one unit, frequency and its TWU. In this part, items are sorted in ascending order of their frequencies. The transactional array is a set of rows in which each row is associated with one item in the index part. Each element in the transactional array stores 4 data fields : quantity in transaction, TU of transaction and location [row, column] of the next item in this transaction. If that item is the last element in a transaction, its location should be empty.

In the first scanning of the database, we calculate transaction utility, total quantity, frequency, TWU of each item. Sort the items in ascending order of frequencies and build the index part.

During the second scanning of the database, we sort each transaction, in ascending order of item's frequency and put it into transactional array as follows:

Based on the Index part we determine the position of the first item. Then in the item's row, we find the first empty place (cell) and save item's information here. Then the location of next item is determined in exactly the same way as it was done for the first item (note that this location will be stored in the cell of the first item). We repeat this for all items in the transaction, in the cell of the last item the location fields are empty.

This transactional array is constructed and saved in the external memory.

The Tables below are used for the demonstration of our algorithm.

To give an example, suppose we have a database in Table 1 and 2, threshold = 30% (of total utility), and $minutil$ = 30% x 398 = 119,4.

The algorithm scans the database for the first time, calculates the transaction utility (in Table 3), total quantity, frequency, and TWU of each item (in Table 4). It then sorts items in ascending order of frequencies and builds the index part (Table 5).

The algorithm then scans the database for the second time, and for each transaction, it sorts the items into ascending order of frequencies and put them into the transactional array. Table 6 illustrates the sorted transactions. In case of transaction $T1 = (B : 12, C : 2, E : 2)$ the search in the index part gives that B is in position 3, C is in position 4 and E is in position 5, so the 3 blocks which are used to save the information of these items are in row 3, row 4 and row 5 of the transactional array. First we find the first empty block in row 3 and we have [3,1], this block will save B's information and the address (or location) of C as well. The first

Table 3: Utility of transactions of database in table 1 and 2

TID	A	B	C	D	E	*tu*
*T*1	0	12	2	0	2	*72*
*T*2	0	12	0	2	1	*71*
*T*3	2	0	1	0	1	*12*
*T*4	1	0	0	2	1	*14*
*T*5	0	0	4	0	2	*14*
*T*6	1	2	0	0	0	*13*
*T*7	0	20	0	2	1	*111*
*T*8	3	0	25	6	1	*57*
*T*9	1	2	0	0	0	*13*
*T*10	0	0	16	0	1	*21*
Sum	8	48	48	12	10	398

Table 4: Quantity, twu and Frequency of items

Item	Quantity	Frequency	twu
A	8	5	109
B	48	5	280
C	48	5	176
D	12	4	253
E	10	8	372

Table 5: Index part of transactional array

Pos	Item	Quantity	Profit/Unit	Frequency	Twu
1	D	12	3	4	253
2	A	8	3	5	109
3	B	48	5	5	280
4	C	48	1	5	176
5	E	10	5	8	372

empty block in row 4 is [4,1] so [3,1] will contain the following parts: 12 (quantity of B), 72 (Transaction Utility), [4,1] (Address of next item in transaction). Then the same process is applied to C and E. Since E is the last item, the location field of E will be empty. We repeat that for all other transactions to obtain the final transactional array given in Table 7.

All needed information from Table 1 and 2 has been transformed into the trans-

Table 6: Sorted transaction in order of frequency.

TID	D	A	B	C	E	tu
$T1$	0	0	12	2	2	*72*
$T2$	2	0	12	0	1	*71*
$T3$	0	2	0	1	1	*12*
$T4$	2	1	0	0	1	*14*
$T5$	0	0	0	4	2	*14*
$T6$	0	1	2	0	0	*13*
$T7$	2	0	20	0	1	*111*
$T8$	6	3	0	25	1	*57*
$T9$	0	1	2	0	0	*13*
$T10$	0	0	0	16	1	*21*

actional array so that we can use this array for mining high utility itemsets (even with different threshold).

Table 7: Transactional array of table 1 and 2

	Transactional Array									
Pos	Index	1	2	3	4	5	6	7	8	9
1	D, 12, 3 4, 253	2, 71 [3,2]	2, 14 [2,2]	2, 111 [3,4]	6, 57 [2,4]					
2	A, 8, 3 5, 109	2, 12 [4,2]	1,14 [5,4]	1,13 [3,3]	3, 57 [4,4]	1, 13 [3,5]				
3	B, 48, 5 5, 280	12, 72 [4,1]	12, 71 [5,2]	2, 13 [∅, ∅]	20,111 [5,6]	2, 13 [∅, ∅]				
4	C, 48, 1 5, 176	2, 72 [5,1]	1, 12 [5,3]	4, 14 [5,5]	25, 57 [5,7]	16, 21 [5,8]				
5	E, 10, 5 8, 372	2, 72 [∅, ∅]	1, 71 [∅, ∅]	1, 12 [∅, ∅]	1,14 [∅, ∅]	2, 14 [∅, ∅]	1, 111 [∅, ∅]	1, 57 [∅, ∅]	1, 21 [∅, ∅]	

The following is our algorithm for building transactional array:

Algorithm 1. (Build transactional array).

Input: Database DB.

Output: Transactional array in external memory.

Method:

1. for each $T \in DB$ // *First time scanning database*
2. begin
3. - Calculate transaction utility $tu(T)$;
4. - Calculate frequency, quantity, TWU of each item;
5. end;
6. Sort all items in ascending order of frequency;

7. Based on sorted items list, build index part of transactional array;
8. for each $T \in DB$ // *Second time scanning database*
9. begin
10. Sort items in T in order of index part, we have following list:
$Tlist = (A_1 : s_1, A_2 : s_2, ..., A_k : s_k)$;
// s_i *is quantity of item in transaction* T.
11. Determine address $[d_1, c_1]$ to save information of item A_1 in transactional array;
12. for i:=1 to $k - 1$ do // *With each item in TList* .
13. begin
14. - Determine address $[d_{i+1}, c_{i+1}]$ where save information of A_{i+1};
15. - Save at $[d_i, c_i]$:Quantity s_i ,Transaction Utility $tu(T)$, Address $[d_{i+1}, c_{i+1}]$;
16. end;
17. Save at $[d_k, c_k]$: Quantity s_k , Transaction Utility $tu(T)$, empty address $[\emptyset, \emptyset]$;
18. end;

4.2 Mining transactional array

Consider all items of the index part of transactional array (top down). For each item i_p, if $TWU(i_p) \geq minutil$ the algorithm gets all transactions from the transactional array that contains that item. From these transactions, it builds the COUI-tree for that item and mines that tree for high utility itemset. It then discards the tree as soon as it has been mined and moves to the next item. COUI-tree of item x must have x as its root. Each COUI-tree has a header table that contains three data fields: item's name, TWU and pointer (pointing to the first and same item in COUI-tree). Each node of COUI-tree includes 4 data fields: item's name, TWU (Utility of transaction that it's inside), an array of quantity of all items from this node up to the root, pointers pointing to the next same label node or null if not. Each transaction is read and inserted into COUI-tree as follow:

Let $[x \mid L]$ be a transaction, where x is the first item and L is the rest of the transactions. The algorithm checks whether item x is one of child nodes of the root. If it is, then update the information for that node correspondingly, otherwise, add a new node as a child of root and labels it x. Consider the present node as the root, repeat the process on the next item in L if it is not empty. When adding a new node, an update of horizontal link of the corresponding item in header table is needed.

The COUI-tree building process is illustrated by an example with the transactional array in Table 7.

To mine the high utility itemsets on the transactional array in Table 7 we need to build COUI-tree for the items: D, B and C. We call COUI-tree corresponding for each item as D-COUI-tree, B-COUI-tree and C-COUI-tree respectively. It is not necessary to build A-COUI-tree since $twu(A) = 109 < minutil$, not for E-COUI-tree because there is only one node (E as root) in this tree. D-COUI-tree contains

all items co-occurring with D in the transactions. B-COUI-tree contains all items co-occuring with B in the transactions except for D and A. C-COUI-tree contains all items co-occuring with C in the transactions except for D, B and A.

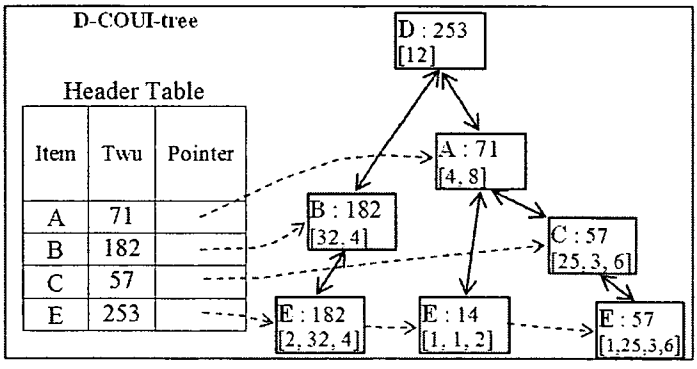
- Building process of D-COUI-tree:

From the index part we know that D's frequency is 4, so there are 4 transactions that contains D inside. Start at the first block in row 1 of transactional array, read information in this block and the address saved in the Location field to reach the next item. Here we get the following sequence.

Starting at [1, 1] we get item D with quantity of 2, this block refers to [3, 2]. At [3, 2] we get item B with quantity of 12, this block refers to [5, 2]. At [5, 2] we get item E with quantity of 1 and an empty Location field so the algorithm stops at this point.

At this first link we get the first transaction of D, $T_1 = (D:2, B:12, E:1)$ and $tu(T_1) = 71$. Likewise, we could get all D's transactions and have: $T_4 = (D:2, A:1, E:1)$ with $tu(T_4) = 14$, $T_7 = (D:2, B:20, E:1)$ with $tu(T_7) = 111$, and $T_8 = (D:6, A:3, C:25, E:1)$ with $tu(T_8) = 57$. Each transaction is read and inserted into D-COUI-tree. It is noted that twu of the header table needs to be adjusted correctly. Figure 1 shows the D-COUI-tree.

Figure 1: D-COUI-tree



- Mining D-COUI-tree:

Mining D-COUI-tree is to find all high utility itemsets that contain D inside. In D-COUI-tree, twu of item A and C is smaller than minutil, so the itemsets that contain them cannot be high utility itemsets and in the candidate generating process we do not generate candidates containing these items.

In turn, consider all items in the header table but this time we do it bottom up, therefore, E will be the first item we encounter. From the pointer in the header table of item E we find 3 nodes in D-COUI-tree labeled E. In the path from the first E to the root we will have (E:2, B:32, D:4) with twu = 182; Push it and all its subsets plus D into $D - list$ (a list contains all high utility candidates containing D) and we will have:

$D - List = \{(E:2, B:32, D:4):182; (E:2, D:4):182; (B:32, D:4):182\}$.

Adjust twu and the array of quantity of each node E, B and D on that path. Twu is subtracted to 182 and the array of quantity is subtracted corresponding (step 1).

The path to root of the second E (E:1, A:1, D:2) with twu = 14 does not generate any candidate that contains A, so only (E:1, D:2) is pushed into $D - List$. In $D - List$, (E:1, D:2) has twu = 182 so this value will be adjusted to 196; Adjust all the values for items E, A and D (step 2).

Likewise, for the third E and we add (E:1, D:6):57 into $D - List$. At this point we have done with item E and move to the next item in the header table. The next items in the header table are C, B and A but all of them have twu = 0 so there is no need to generate any candidates from them. Figure 2 shows this process.

Finish mining D-COUI-tree we have a candidate list in $D - List$. Traverses all candidates and with each $X \in D - List$, we calculate actual utility, if $u(X) \geq minutil$ then X is a high utility itemset. With D-COUI-tree we find $HU = \{EBD(182), \text{BD}(172)\}$. Repeat this process for the next items, in the end, the result will be:

$$HU = \{EBD(182), \text{BD}(172), EB(240), B(240)\}.$$

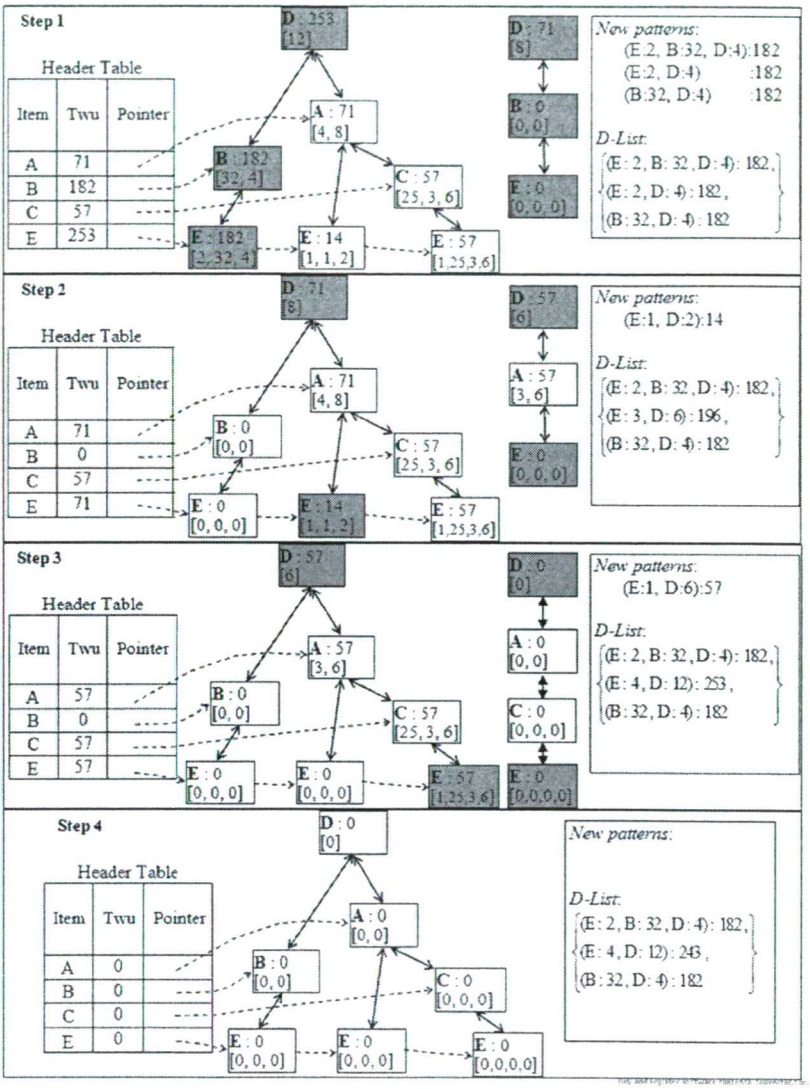
We can describe the algorithm for building and mining COUI-tree as follows

Algorithm 2 (Build and mine COUI-tree).
Input: Transactional array, utility function, threshold *minutil*.
Output: HU set contains all high utility itemsets of database DB.
Method:

1. From top down, A = first item in index part satisfy $twu(A) \geq minutil$;
2. **repeat**
3. **if** $twu(A) < minutil$ **then** goto 15;
4. Calculate utility of A; // *base on quantity and unit utility.*
5. **if** $u(A) > minutil$ **then** $HU := HU \cup \{A\}$;
6. Read frequency s and location of row d that contains A in transaction;
7. Create root R with label A of (A)-COUI-tree, assign twu = 0, quantity = 0;
8. **for** i:=1 **to** s **do** //*traverse all s blocks in row d of transactional array;*
9. **begin**
10. - Start at [d, i], determine $T = (A_1 : s_1, A_2 : s_2, , A_k : s_k)$ and transaction utility $tu(T)$; //A_1 *is item A.*
11. - Call *insert_tree*(T, R) function to insert T into (A)-COUI-tree;
12. **end**;
13. Call MineCOUI-tree (A); // a function mines (A)-COUI-tree.
14. Free (A)-COUI-tree;
15. A = next item in index part;
16. **Until** (A is the last one in index part);
17. Calculate utility of A;
18. **if** $u(A) > minutil$ **then** $HU := HU \cup \{A\}$;

Here is MineCOUI-Tree function.

Figure 2: 4 steps to mining D-COUI-tree



Function: MineCOUI-tree (A);
Method:
1. $(A) - List := \emptyset$ *//Initialize the empty candidate list.*
2. **for each** (item B in header table of (A)-COUI-tree) *//bottom up.*
3. **for each** (*node N on (A)-COUI-tree that labeled B*)
//follow the pointer from header table.
4. **begin**

5. - Read twu and quantity array of all items of node N;
6. - Determine pattern X in the path from N up to the root;
7. - Generate subset of X that contains A *//discard all low utility TWU items.*
8. - Push all subsets generated above into $(A) - List$;
9. - Adjust twu and array of quantity of all items on the path of N to the root;
10. **end**; *//Finish mining* $(A) - COUI - tree$.
11. **for each** $Y \in (A) - List$ *// Traverse all candidates in* $(A) - List$.
12. **begin**
13. - Calculate utility $u(Y)$ of candidate Y;
14. - **if** $u(Y) > minutil$ **then** $HU := HU \cup \{Y\}$;
15. **end**;
16. Return HU;

5 Algorithm Evaluation and Performance Study

5.1 Algorithm Evaluation

a) Algorithm 1: Transaction Array construction

+ Pass I:

- Calculation of transaction utility $tu(T)$, calculation of frequency, quantity, TWU of each item. Hence, the total time complexity of this step is $O(n)$.

- Sorting of all items in ascending order of frequency costs $O(n.logn)$ in time.

- Based on the sorted item list, the building index part of transactional array has time complexity of $O(n)$.

+ Pass II: For each T of DB, we need to identify $Tlist = (A_1 : S_1, A_2 : S_2, ..., A_k : S_k)$, S_i is number of A_i item in transaction T. With each item in $TList$, address $[d_{i+1}, c_{i+1}]$ where save information of A_{i+1} needs to be determined, that makes the total time complexity of $O(n^2)$. In summary, the time complexity for algorithm 1 is $O(n^2)$.

b) Algorithm 2: Mining the transaction array.

+ Building the COUI-tree: At the turn of a top-down data items, time complexity for tree construction of algorithm is $O(n^2)$. Since there are n data items, the total time complexity to build all trees COUI-tree is $O(n^3)$.

+ Minning the COUI-tree: Algorithms considers in turn each data item in the header table, with each B data item to browse nodes in COUI-tree labeled B. Suppose the height of the tree is h, to generates any candidate patterns then generated sub patterns , need a running time complexity of 2^{h-1}.

The greatest height of COUI-trees is equal to the length of the longest transaction in the database transaction: $max(h) = max\{|T|, T \in DB\}, 1 \leq h \leq n$. In the worst case, the database have transactions that include all items, $max\{|T|, T \in DB\} = n$, so $max(h) = n$. In that case, time complexity to mine highest COUI-tree is $O(2^{n-1})$, thefore, the time complexiyty for algorithm 2 is $O(2^n)$.

Algorithm time complexity is the total time complexity of algorithm 1 and algorithm 2 making it as $O(2^n)$ (n is the number of data items).

Although in theory the worst case time complexity of the algorithm is $O(2^n)$, in reality, transactions databases are often extremely sparse, the height h of the tree COUI-tree could be very small compared to n, so the practical running of the algorithm often does not suffer from combinatorial explosion.

5.2 Performance Study

The algorithm was written in Microsoft Visual C ++ 6.0, running on a PC with a Pentium dual core 2.0 GHz CPU, 1 GB of RAM, using Windows XP Professional operating system. The program reads data from files and outputs to a data file. The algorithm was experimented on several real and synthetic data sets. Retail is a market basket dataset from a Belgian supermarket (Brijs, Goethals, Swinnen, Vanhoof, & Wets, 1999). Retail transaction file contains 88,162 transactions, 16,470 items and the average length of transactions is 10.31 [5]

We generated two synthetic datasets using our own program and IBM Quest data generator [10]: (a) T10I500D100K, the average length of transactions is 10.74, with 500 items and the number of transactions is 100K, (b) 10I1000D100K, the average length of transactions is 10.10, with 1000 items and the number of transactions is 100K.

Table 8 shows the characteristics of the datasets. Since all these datasets are normally used for testing traditional frequent itemset mining algorithms, we added quantity and item utility values to the dataset. We generated a utility table based on lognormal distribution with the utility values ranging from 0.1 to 10. The quantities of items were generated randomly in the range of 1 to 10. Test results are shown in Figure 3.

Transactional data is converted into a new database layout set in the external memory, so the algorithm can mine very large datasets. Running time of COUI-Mine algorithm includes data conversion time for the transaction array and the mining of transaction array. Once data has been converted into a new database layout, it can be mined with different utility thresholds without converting the data, hence, running time of the algorithm is reduced to the time to mine the transaction array only. On the dataset Retail, the data conversion time was 4744 seconds. Table 9 shows the running time of the algorithm on dataset Retail with different utility thresholds.

6 Conclusion

Based on the results of the experiments and analyses of the algorithm, some conclusions could be drawn as follow:

+ It needs to be scanned database twice to build transactional array and this array contains enough information for mining high utility itemsets. This transactional array is stored in the external memory, so the algorithm can mine very large

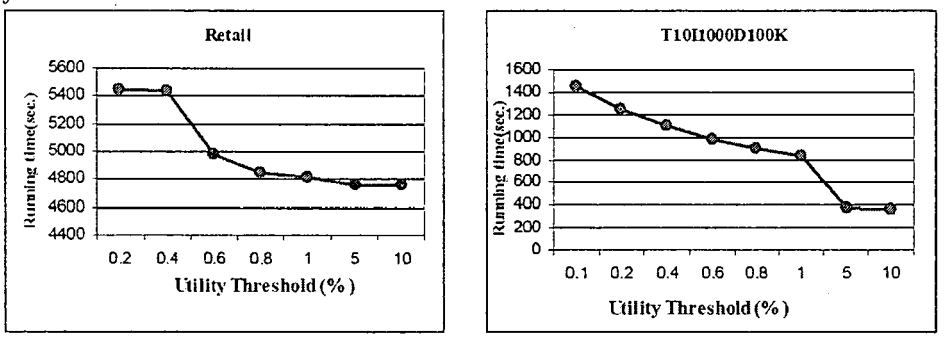
Table 8: Characteristics of Datasets

Dataset	Number of transactions	Number of Items	Average Length
Retail	88.162	16.470	10,31
T10I500D100K	100.000	500	10,74
T10I1000D100K	100.000	1000	10,10

Table 9: Execution time on dataset Retail

Utility Threshold	COUI-Mine		
	Phase 1	Phase 2	Total
0,2	4744	702	5446
0,4		697	5411
0,6		237	4981
0,8		99	4843
1		76	4820
5		12	4756
10		11	4755

Figure 3: Execution time with varying minimum utility thresholds on real and synthetic datasets



databases.

+ Mining transactional array is based on small structure of COUI-tree. At each time, only one tree is in the memory, it means that we only store in the memory a small part of the data. Otherwise, mining COUI-tree is using non-recursive algorithm so it reduces time and memory needed in the mining process.

+ After the transactional array is built, the algorithm can mine with arbitrary thresholds.

+ The algorithm avoids massive computations because it does not need to generate candidates and check for constraints like in some other approaches.

+ The algorithm also uses the concept of TWU effectively to reduce the time complexity to generate candidates.

In conclusion, COUI-Mine is an effective algorithm for mining high utility itemsets in large datasets.

References

[1] Nguyen Huy Duc, *"Mining Association Rule in Large Databases"*, In Proceeding of the First National Symposium Fundamental and Applied Information Technology Research (FAIR), Ha Noi, 2003.

[2] Nguyen Thanh Tung, *"Mining High Utility Itemsets in Databases"*. Journal of Computer Science and Cybernetics, Viet Nam, vol. 23, no. 4, pp. 364-373, 2007.

[3] Vu Duc Thi and Nguyen Huy Duc, *"Efficient Algorithm for Mining High Utility Itemsets Based On Prefix-trees"*, Journal of Computer Science and Cybernetics, Viet Nam, vol. 24, no. 3, pp. 204-216, 2008.

[4] R. Agrawal and R. Srikant, *"Fast algorithms for mining association rules"*. In pro-ceedings of 20^{th} International Conference on Very Large Databases, Santiago, Chile, 1994.

[5] Frequent Itemset Mining Implementations Repository, 2003. `http://fimi.cs.helsinki.fi/data/`

[6] H. Yao and H. J. Hamilton, *"Mining itemset utilities from transaction databases"*. Data & Knowledge Engineering, vol. 59, pp. 603- 626, 2006.

[7] Y. Liu, W.-K. Liao, and A. Choudhary, *"A Fast High Utility Itemsets Mining Algorithm"*, Proc. UBDM'05, Chicago Illinois, 2005.

[8] M. El-Hajj and Osmar R. Zaiane. *"COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation"*. In Proc. 2003 Intl Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD), August 2003.

[9] M. El-Hajj and Osmar R. Zaiane. *"Inverted matrix: Efcient discovery of frequent items in large datasets in the context of interactive mining"*. In Proc. 2003 Intl Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD), pp. 109-118, August 2003.

[10] IBM Synthetic Data Generator, `http://www.almaden.ibm.com/software/quest/resources/index.html`

Received 18th May 2009

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János